UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

DESARROLLO DE UN PROGRAMA DE GENERACIÓN DE CARGAS COMPUTACIONALES

Autor: Juan Francisco Canseco Moreno-Palanca

Tutor: David Expósito Singh

Septiembre de 2012

AGRADECIMIENTOS

En primer lugar, quiero dar las gracias a mi familia, ya que siempre ha sido y seguro que será un gran punto de apoyo para mí en el futuro. Especialmente agradecido estoy a mis abuelos, Francisco y Lola, a mi padre, Juan Francisco, mi hermana, Paula, mis tíos, Fernando y Gema, y a mi novia, Nines, por toda la confianza que siempre habéis tenido en mí, por todo vuestro apoyo y por hacer de mí la persona que hoy en día soy. Estoy muy agradecido a cada uno de vosotros y espero algún día poder recompensaros todo lo que dais y me habéis dado en la vida.

También quiero acordarme de todas aquellas personas que han recorrido conmigo este camino, no solo en la Universidad, sino desde que empecé a dar la lata con 3 años en el Colegio Francisco de Quevedo, pasando posteriormente por el IES Enrique Tierno Galván de Leganés. Me gustaría dar las gracias a compañeros y profesores, ya que de todos o de la gran mayoría, guardo un gran recuerdo.

Muchas gracias a los compañeros con los que he compartido estos cuatro años en la Universidad, con los que he sufrido, pero con los que sobre todo he reído y me han hecho pasar momentos inolvidables. Muchas gracias a Christian, Sergio, Adrián, José Ángel, Javi, Roberto, Tamara, Ignacio, Rodrigo y todos aquellos con los que he coincidido.

Por último, quiero dar las gracias de forma especial a mi tutor, David, por su paciencia, su cercanía y su ayuda durante la realización de este proyecto, ya que ha hecho parecer todo mucho más fácil, siempre con palabras de ánimo y viendo las cosas de forma positiva.

Muchas gracias a todos

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 – INTRODUCCIÓN

1.1 – MOTIVACIÓN Y PROBLEMÁTICA

En la actualidad, el Cloud Computing es una arquitectura que es muy demandada por los usuarios, por la multitud de ventajas que brinda a los usuarios. Entre esos servicios se encuentra el servicio conocido como *pay-as-you-go* [1], generalmente utilizado para el suministro de recursos computacionales, ya sea para almacenamiento o para procesamiento de datos.

El problema radica en que los usuarios no tienen un acceso exclusivo a los datos, y todos éstos deben compartir los recursos del Cloud. Debido a esto podemos decir que el rendimiento es impredecible, ya que una máquina virtual que hace un uso excesivo, por ejemplo, de la entrada/salida, afecta al rendimiento del resto de las aplicaciones que estén ejecutándose en el sistema.

El proyecto surge en el contexto de la línea de investigación de ejecución de altas prestaciones para el Cloud Computing, buscando el desarrollo de técnicas que permitan ejecutar de forma eficiente aplicaciones paralelas en este tipo de arquitectura.

La motivación que esta problemática crea es la de la creación de un entorno de ejecución capaz de generar diferentes cambios en el rendimiento de la plataforma. La forma en la que se generará dicho escenario será mediante la creación de un software que permita la generación de diferentes cargas computacionales de forma controlada, realizando operaciones de red, de CPU y de acceso a memoria en el sistema.

A través de esta herramienta es posible desarrollar técnicas en el grupo de investigación que permitan a los programas adaptarse a estos cambios en las condiciones de ejecución de los mismos.

[1] Pago por consumo

1.2 – OBJETIVOS

Una vez expuesta la motivación que nos lleva a la realización del proyecto, se enumeran una serie de objetivos que se desean cumplir al fin del mismo.

El principal objetivo es el de desarrollar una aplicación paralela que permita introducir de forma artificial condiciones de variabilidad en el entorno de ejecución. Estas condiciones se generarán introduciendo de forma controlada diferentes cargas computacionales, capaces de variar el tráfico de red, la carga de la CPU y el tráfico de memoria de cada nodo, dependiendo de las condiciones locales del mismo y del enlace de red. La introducción de estas cargas generará una serie de interferencias, y alterará el rendimiento de los demás programas. De este modo, se busca alterar de forma controlada el rendimiento de los programas ejecutados en un entorno de Cloud.

Para superar el objetivo principal, tenemos una serie de objetivos específicos, como:

- Familiarización con la interfaz de paso de mensajes MPI.
- Diseño de la aplicación que permita generar cargas computacionales.
- Evaluación individual de la aplicación aislada.
- Evaluación del efecto de la aplicación en un entorno real, estudiando el rendimiento de otras aplicaciones paralelas.

1.3 - ESTRUCTURA DEL DOCUMENTO

En esta sección se ofrece un esquema del documento, junto a una breve descripción del contenido de cada apartado:

Capítulo 1: Introducción

Este apartado servirá como presentación de aquello que se quiere desarrollar, las motivaciones que han llevado a ello, y una serie de objetivos que se desean cumplir.

Capítulo 2: Estado de la cuestión

En este apartado se hace una descripción del estado de las tecnologías relevantes en la época del desarrollo del trabajo, comentando las propiedades o características más importantes de las mismas.

Capítulo 3: Descripción del Sistema

Se realiza un análisis en profundidad de los requisitos del sistema a desarrollar y se define la solución, así como el entorno de desarrollo utilizado en el proyecto.

Capítulo 4: Gestión del Proyecto

Se describen los aspectos relacionados con la gestión del proyecto y la planificación del mismo.

Capítulo 5: Pruebas

Se ofrece una descripción de las diferentes plataformas en las que se realizarán las pruebas, una especificación de las pruebas a realizar sobre la aplicación y el resultado de las mismas.

Capítulo 6: Evaluación junto a Aplicación Paralela

Se ofrece una descripción de la aplicación paralela utilizada para la realización de la evaluación, una especificación de las pruebas a realizar y los resultados de las pruebas realizadas en un entorno real.

Capítulo 7: Presupuesto

Se realiza un cálculo del coste que ha supuesto la realización del proyecto, detallando todas las fases y factores de costes.

Capítulo 8: Conclusiones y trabajos futuros

Se exponen las conclusiones extraídas tras la realización del proyecto y posibles trabajos o estudios futuros.

Capítulo 9: Referencias

Listado de referencias utilizadas para la realización del proyecto.

Anexos

Apartado en el que se incluirá información extra como manuales de usuario o ejemplos de la ejecución de la aplicación desarrollada.

2 – ESTADO DE LA CUESTIÓN

En este apartado se realiza un pequeño resumen de aquellas tecnologías relacionadas con el desarrollo del proyecto.

2.1 - CLUSTER COMPUTING

Cuando se habla de un *clúster* de computadores, se habla de un grupo o conjunto de computadores, en el que no es necesario que todos sus nodos dispongan de un mismo hardware o SSOO[2], unidos mediante una red de alta velocidad, que se comportan de la misma forma que si se tratara de un único computador.

Esta arquitectura ha de estar gestionada por un sistema que se encargue de interactuar con el usuario y los procesos que en él se ejecutan para optimizar el rendimiento.

El concepto de *clúster* surge a partir de las actuales tendencias en las que se incluye la disponibilidad de microprocesadores de alto rendimiento y las crecientes necesidades de una gran potencia de cómputo para aquellas aplicaciones que la requieran. Económicamente, se trata de una arquitectura muy atractiva, ya que es más viable la adquisición de varios computadores de unas prestaciones medias para después conjuntarlos en un único y potente sistema que la de obtener un solo computador con grandes recursos hardware.

Las características principales que se deben tener en cuenta a la hora de hablar de un clúster son:

- Alto rendimiento
- Alta disponibilidad
- Balanceo de Carga
- Escalabilidad

Han pasado a formar una parte importante en el avance de industrias como la de investigación científica y de defensa, procesos de energía sísmica y otras áreas específicas destacadas como la investigación del genoma humano y curas para enfermedades como el SIDA.

[2] Sistema operativo.

2.2 – CLOUD COMPUTING

El concepto de Cloud Computing se refiere tanto a las aplicaciones utilizadas como servicios a través de Internet como al hardware y los sistemas de software de los centros de datos que proporcionan dichos servicios. De esta forma, se definen dos términos fundamentales en el desarrollo de la computación en la nube. Por un lado tendríamos el *SaaS (Software as a Service)* que corresponde a los servicios prestados, y por otro el centro de datos de hardware y software conocido como una nube.

Cuando una nube se pone a disposición pública en un proceso conocido como *"pago por consumo"*, se denomina como nube pública y generalmente se utiliza para la venta de servicios de *utility computing.* Denominamos a un servicio de *utility computing* cuando suministra recursos computacionales, como pueden ser el almacenamiento y procesamiento de datos. Un ejemplo actual de una nube pública puede ser el servicio web de Google. En cambio, se utiliza el término de nube privada para referirse a los centros de datos internos de una empresa u otra organización que no esté a disposición del público. Así pues, el Cloud Computing puede definirse como la suma del *SaaS* y la nube pública, quedando normalmente a un lado las nubes privadas. En la ilustración 1 se muestran las funciones de los usuarios o proveedores que se diferencian en capas de Cloud Computing. En un primer nivel aparece el proveedor del Cloud, es decir, el dueño de los recursos, que puede vender a un tercero para que éste lo utilice o provea a su vez un servicio a otros usuarios, como por ejemplo podría ser una Aplicación Web.

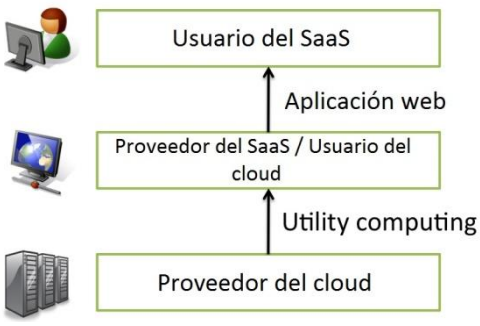


Ilustración 1: Capas de Cloud Computing

Analizando el Cloud Computing desde la perspectiva del hardware, son tres las novedades más reseñables que se observan:

- Se crea la ilusión de una infinidad de recursos informáticos disponibles para cubrir la demanda de los usuarios finales. De esta forma se elimina la necesidad de que la nube pública necesite anticiparse al aprovisionamiento de recursos según la cantidad de demandas.

- Elimina el compromiso de contratación de recursos por adelantado por parte de los usuarios de la nube. Esto permite a las empresas comenzar poco a poco y aumentar sus recursos hardware cuando las necesidades así lo exijan.

- Se añade la capacidad de realizar un pago por el uso de los recursos informáticos a corto plazo (el almacenamiento por día, por ejemplo). Al mismo tiempo, se podrá realizar un alquiler de los mismos o almacenarlos para eliminar su utilización según su utilización.

2.3 – BENCHMARKS

El benchmarking es una técnica muy utilizada para medir el rendimiento de un sistema, o de un componente del mismo, con el objetivo de después poder compararlo con los de otros sistemas. Esta técnica puede ser utilizada para medir el rendimiento de diferentes componentes (como CPU, memoria RAM…) o de un software, o parte del mismo. En cuanto a los componentes, también puede estar dirigido al análisis de una función específica dentro de cada uno de ellos.

Originalmente se utilizaban para estimar el tiempo de ejecución de un programa, pero con el avance de las tecnologías se ha convertido en una técnica muy extendida a la hora de evaluar las prestaciones de una plataforma.

Un ejemplo de los benchmarks más importantes es el SPEC (Standard Performance Evaluation Corporation), que es un consorcio en el que se incluyen multitud de vendedores de computadores, grupos de investigación, universidades e integradores de sistemas de todo el mundo, con el objetivo principal de crear un estándar para medir el rendimiento de los equipos, y después analizar y publicar los resultados.

Ilustración 2: Logotipo de SPEC

Se incluye esta técnica en el documento ya que la aplicación que se desarrollará en este proyecto puede ser considerado un benchmark, aunque no en su propósito principal, ya que el desarrollo de éste no buscará medir datos específicos de otras aplicaciones, sino variar las condiciones locales de ejecución de esas aplicaciones.

2.4 – MPI (INTERFAZ DE PASO DE MENSAJES)

MPI es un estándar diseñado para ser utilizado en programas que exploten la existencia de múltiples procesadores, en el que se define tanto la sintaxis como la semántica utilizada en las funciones de la biblioteca.

La técnica de paso de mensajes es empleada en la programación concurrente para permitir la exclusión mutua, así como la sincronización de procesos. Es muy importante en la programación de sistemas distribuidos, pues no necesita memoria compartida.

El protocolo define tres elementos que serán los principales actores que intervendrán en la comunicación.

- Proceso emisor del mensaje
- Proceso receptor del mensaje
- Mensaje a enviar

La forma de enviar los mensajes es importante, pues se puede realizar de forma síncrona o asíncrona. En el paso de mensajes asíncrono, el proceso emisor no se preocupará de que el proceso receptor reciba el mensaje, y solamente se encargará de enviar el mensaje, pudiendo después continuar con su ejecución. En el caso de las llamadas síncronas, el proceso emisor esperará a que el proceso receptor capte el mensaje para continuar con su ejecución. Un ejemplo de aplicación de las llamadas síncronas son las llamadas a procedimientos remotos, muy utilizadas y extendidas en las arquitecturas cliente/servidor.

Este protocolo, de comunicación entre distintas computadoras, ofrece importantes ventajas sobre otras bibliotecas de paso de mensajes, gracias a que es portable, y muy rápida, debido a que cada implementación está optimizada para el hardware en el que será ejecutado.

Como características principales, destacan las siguientes:

- Cada proceso es identificado por un una variable denominada *rank,* que permite determinar qué proceso ejecuta ciertas porciones de código.
- Se define un comunicador, como una colección de procesos que pueden comunicarse entre sí. El comunicador básico permite la comunicación entre todos los procesos activos durante una ejecución.

- A partir del estándar de MPI 2.0, se definen una serie de funcionalidades extra, entre las que destacan:
    - Gestión dinámica de procesos, que permite a un proceso MPI tanto la creación de nuevos procesos MPI como establecer una comunicación con otros procesos MPI creados por separado. Anteriormente el número de procesos a ejecutar debía ser asignado antes del lanzamiento del programa, no siendo posible la creación de procesos adicionales durante la ejecución.
    - Funciones para la gestión de Entrada/Salida paralela en sistemas distribuidos, permitiendo el acceso a los archivos de una forma más sencilla. Debido a la escasa investigación realizada, se trata de una característica que no ha obtenido de momento un gran rendimiento.

Las diferentes llamadas de MPI pueden clasificarse en cuatro subgrupos:

1. Llamadas para inicializar, administrar y finalizar comunicaciones.
2. Llamadas para transferir datos entre un par de procesos (llamadas punto a punto).
3. Llamadas para transferencia de datos entre varios procesos (llamadas colectivas).
4. Llamadas para crear tipos de datos definidos por el usuario.

Una vez comentadas las principales características del estándar MPI, cabe destacar que es un estándar ampliamente utilizado en la programación de aplicaciones de Cluster Computing, gracias a que es idónea para trabajar en entornos distribuidos.

3 – DESCRIPCIÓN DEL SISTEMA

En este punto del documento se hará una descripción de la arquitectura que se implementará, el entorno necesario para su realización y se realizará un análisis de los requisitos que la aplicación deberá cumplir.

3.1 – DESCRIPCIÓN DE LA ARQUITECTURA (NIVEL SOFTWARE)

Como ya se ha comentado en puntos anteriores de este documento, con la realización de este proyecto se desarrollará una aplicación que debe de ser capaz de generar una serie de cargas computacionales que causen una interferencia real en la ejecución de otros procesos, con el objetivo de estudiar la importancia que esas interferencias tienen en el tiempo de ejecución de los procesos que corren en su misma máquina.

La aplicación, denominada Generador_Carga, será una aplicación MPI, que como se ha comentado en anteriores apartados, permite la explotación de múltiples procesadores para la ejecución de tareas. Gracias a esto, la aplicación MPI nos dará la posibilidad de poder replicar procesos, pudiendo éstos comunicarse entre sí para la realización de diferentes trabajos, etc. El número de procesos creados dependerá de la parametrización introducida en la llamada al programa.

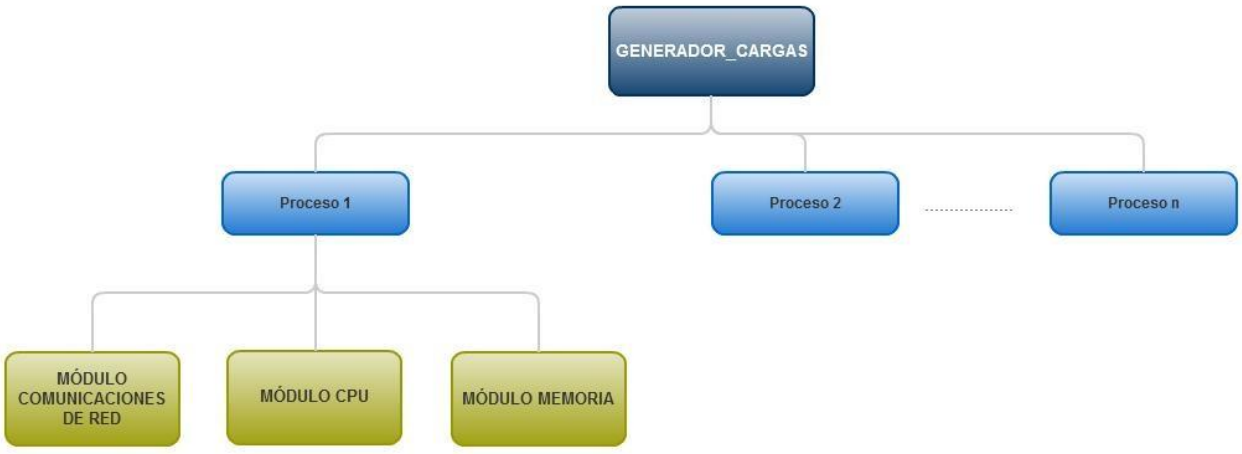


Ilustración 3: Diseño de la aplicación Generador_Carga

Como podemos ver en la Ilustración 3: Diseño de la aplicación, cada proceso creará varios hilos de ejecución o threads (figuras amarillas), que se corresponden con los diferentes módulos de la aplicación. Estos módulos introducirán de forma controlada cargas computacionales con el objetivo de variar las condiciones locales de ejecución en los diferentes nodos y la red.

Los diferentes módulos que contiene la aplicación son los siguientes:

- *Módulo de Comunicaciones de red*: Se generará tráfico de red entre un par o un conjunto genérico de nodos.
- *Módulo de CPU*: Se producirá un aumento en la utilización de la CPU del nodo al introducir operaciones en el procesador que consuman tiempo de cómputo.
- *Módulo de Memoria*: Se aumentará el tráfico entre el procesador y la memoria en cada nodo del sistema.

A continuación se detallará la arquitectura de cada módulo por separado.

3.1.1 – Módulo de Comunicaciones de red

Este módulo se basará principalmente en la utilización de las llamadas punto a punto de MPI para realizar comunicaciones entre procesos. Estas comunicaciones entre procesos generarán un aumento del tráfico de red del sistema.

Cada proceso MPI, identificado con un *rank,* establecerá las comunicaciones con otros procesos siguiendo el siguiente protocolo:

- En un fichero externo se almacenará una matriz, en el que se definirán las comunicaciones entre los diferentes procesos. Los valores de la matriz se corresponderán con los identificadores de tipo de comunicación definidos. Un ejemplo de matriz para lanzar 5 procesos sería la siguiente:

Procesos	P1	P2	P3	P4	P5
P1	X	1	0	0	3
P2	1	X	0	0	3
P3	0	0	X	2	0
P4	0	0	2	X	0
P5	3	3	0	0	X

Tabla 1: Ejemplo de matriz de comunicaciones

*Los identificadores 1, 2 y 3 indican el tipo de comunicación, mientras que el valor 0 significa que no se realiza comunicación entre los procesos. Además, se considera que los procesos no se comunicarán con sí mismos.

- Cada tipo de comunicación está definido como una estructura C que contiene una serie de atributos, que son:

  - size : Número entero que indica el tamaño del mensaje que se enviará en la comunicación (en bytes).
  - tsend : Timeout de espera para la realización de las comunicaciones en segundos.
  - psend : Número entero que debe tener valor 0 ó 1. Si el valor es 0, el proceso que realizará en envío de los datos será aquel que tenga menor *rank* , mientras que el de mayor *rank* será el que reciba los datos. Por el contrario, si el valor de la propiedad es 1, el proceso con mayor *rank* iniciará la comunicación y el de menor *rank* recibirá el mensaje.

  Un ejemplo de tipo de comunicación sería el empleado por los procesos P1 y P2 de la matriz de ejemplo contenida en la Tabla 1: Ejemplo de matriz de comunicaciones (color rojo):

	Tipo 1	Tipo 2	Tipo 3
Tamaño mensaje	128 bytes	64 bytes	512 bytes
Tiempo espera	3 segundos	2 segundos	5 segundos
Proceso emisor	MIN_RANK (0)	MIN_RANK (0)	MAX_RANK(1)

  Utilizando el tipo de comunicación número 1, el proceso de menor rango (P1) enviaría un mensaje de 128 bytes de tamaño al proceso P2, de mayor rango. Esta comunicación se realizaría cada tres segundos.

- Una vez cargado el archivo en el que se almacena la matriz de comunicaciones, cada proceso leerá aquella fila correspondiente a su *rank*, y elemento a elemento decidirá la acción a realizar. El pseudocódigo es el siguiente:

```
MIENTRAS (1)
   leer_matriz
   tomar_tiempo1 ( función MPI_Wtime() )
```

```
MIENTRAS (elemento != nulo)
SI (tipo_comunicacion != 0 Y rank != myrank)
  MIENTRAS ( tomar_tiempo2-tomar_tiempo1 < tipo_comunicacion.tsend)
      Esperar
  FIN MIENTRAS
  SI (tipo_comunicación.psend = 1 Y myrank > rank)
      Enviar ( función MPI_Send() );
      Medir tiempo ejecución
  FIN SI
SI (tipo_comunicación.psend = 1 Y myrank < rank)
      Recibir ( función MPI_Recv() );
      Medir tiempo ejecución
  FIN SI
  SI (tipo_comunicación.psend = 0 Y myrank < rank)
      Enviar ( función MPI_Send () );
      Medir tiempo ejecución
  FIN SI
  SI (tipo_comunicación.psend = 0 Y myrank > rank)
      Recibir ( función MPI_Recv () );
      Medir tiempo ejecución
  FIN SI
FIN SI
  elemento = elemento + 1
FIN MIENTRAS
FIN MIENTRAS
```

Los procesos se encargarán de leer la matriz de comunicaciones y realizar las acciones posteriores. Como ya se ha comentado, cada proceso únicamente leerá la línea correspondiente a su número de proceso con el objetivo de identificar aquellas comunicaciones en las que dicho proceso participará.

Tras iniciar un contador de tiempo, se realizará una espera para la realización de la comunicación, dependiendo del tiempo establecido en el tipo de comunicación correspondiente. Una vez expirado el *timeout* de espera, cada proceso conocerá mediante el atributo *psend* del tipo de comunicación si debe ser el emisor o el receptor del mensaje. Dependiendo de éste parámetro se realizará la acción que corresponda, haciendo previamente una comprobación del rango del proceso emisor y del receptor, con el objetivo de finalmente realizar las llamadas a las funciones de MPI para el establecimiento de la comunicación. Estas llamadas, serán:

- Para enviar: MPI_Send()
- Para recibir: MPI_Recv()

Una vez realizada la comunicación, se tomará el tiempo transcurrido entre el inicio y el fin de la misma. Esta medida permite la medición del ancho de banda utilizado en la comunicación entre el par de procesos.

3.1.2 – MÓDULO DE CPU

Este módulo tratará, al igual que los demás, de introducir cargas computacionales de forma controlada, y variantes con el tiempo. En este caso, se introducirán operaciones de cómputo en la CPU con el objetivo de hacer una utilización constante de CPU. El porcentaje de uso de la CPU que se utilizará se podrá definir como un parámetro del programa.

Además, el thread recibirá como parámetro de la función un tiempo de timeout, que se aplicará tras realizar un bucle de operaciones para introducir carga al procesador. Posteriormente la función realiza una llamada al sistema, en la que consulta el porcentaje de utilización de CPU del thread. Toda esta operación se realiza una serie de veces, 20 por defecto, para obtener finalmente una media de todos los resultados. Se realiza esta operación con el objetivo de obtener una medida más precisa del porcentaje real de consumo de la CPU.

Una vez obtenido el porcentaje del tiempo de CPU que el hilo está utilizando en el procesador, se realiza una comparación con el valor requerido. Las posibles acciones que se realizarán dependiendo del resultado, serán mostradas en el siguiente pseudocódigo:

```
SI ( porcentaje > (limite + 4)
   timeout = 1.1 * timeout
   cambiado = true
FIN SI
SI ( porcentaje < (limite – 4)
   timeout = 0.9 * timeout
   cambiado = true
FIN SI
SI ( cambiado = true )
   Crear nuevo thread(timeout)
FIN SI
```

Observando el pseudocódigo, se observa que si el porcentaje obtenido está en valores superiores o inferiores a 4 puntos sobre el valor requerido, se creará un nuevo thread.

Para la creación del nuevo thread es necesario recalcular el tiempo que dormirá el thread tras la realización de las operaciones de cómputo. Si el porcentaje supera el límite, el nuevo thread tendrá un timeout superior, con el objetivo de reducir el tiempo de cómputo. En caso contrario, el timeout se reducirá para hacer una mayor utilización de la CPU. En el caso de que el porcentaje de uso de la CPU esté dentro del umbral determinado como aceptable, continuará la ejecución del mismo thread.

El motivo de crear un nuevo thread es que la llamada al sistema en la que se obtiene el porcentaje de uso de la CPU, calcula el mismo haciendo una media desde el inicio de la ejecución del thread. Por esto, cada vez que se modifique el timeout, se creará un nuevo thread hasta que el porcentaje se estabilice en el límite requerido.

3.1.3 – MÓDULO DE MEMORIA

El tercer y último módulo contenido en el programa Generador_Carga introducirá, como los demás casos, cargas en el sistema, aumentando en este caso el tráfico entre el procesador y la memoria.

En primer lugar, debemos conocer los datos de la arquitectura a evaluar. Los datos que hemos de conocer son los siguientes:

- Tamaño de bloque de memoria = Tamaño línea de caché (en bytes).
- Tamaño de palabra en bytes.
- Número de palabras/bloque = $\frac{Tamaño\ de\ bloque}{Tamaño\ de\ palabra}$
- Stride: Se refiere al número de posiciones de memoria existentes entre dos datos consecutivos del vector o objeto donde se almacenan los datos. Generalmente el stride unidad (valor 1) es más eficiente, debido a los efectos que un mayor stride tendría en la caché, por el principio de localidad espacial.

A partir de estos datos, calcularemos los bloques de memoria que serán accedidos a partir de un volumen de datos totales con la siguiente operación:

$$Número\ de\ bloques\ = \left\lceil \frac{Volumen\ de\ datos}{Tamaño\ de\ bloque} \right\rceil \times Stride$$

Multiplicaremos por Stride siempre y cuando éste sea menor que el número de palabras de un bloque de memoria. De ser mayor, la ecuación será la siguiente:

$$Número\ de\ bloques\ = \left\lceil \frac{Volumen\ de\ datos}{Tamaño\ de\ bloque} \right\rceil \times \frac{Número\ de\ palabras}{Bloque}$$

Además, dependiendo de un ancho de banda dado, se calculará el tiempo necesario para realizar la operación (en segundos):

$$Tiempo\ = \frac{Tamaño\ de\ bloque\ \times Número\ de\ bloques}{Ancho\ de\ banda^{3}}$$

Una vez calculados estos datos, propios de la arquitectura, se realizarán operaciones de lectura y escritura de forma continua en un volumen de datos seleccionado. Estas operaciones se podrán regular, ya que es posible introducir como parámetro el porcentaje de lecturas / escrituras que se realizarán.

Además, se pretende fijar un ancho de banda y un tipo de operación para generar un consumo constante de ancho de banda entre la CPU y la memoria.

Para la realización de las operaciones entre la CPU y la memoria se realiza el siguiente algoritmo escrito en pseudocódigo:

```
MIENTRAS ( fin = 0 )
    Reservar buffer en memoria (función malloc)
    Inicializar datos del buffer
```

[3] En bytes/segundo.

```
DESDE i=0 HASTA i=DATOS_TOTALES
     SI ( i % 100 < porcentaje_rw )
          datos = datos + buffer[i] ( lectura)
     SI NO
          buffer[i]=datos ( escritura)
     FIN SI
FIN DESDE
Liberar memoria
Dormir ( 2 segundos )
FIN MIENTRAS
```

Analizando el pseudocódigo, se observa que en primer lugar se realiza la reserva del espacio del volumen de datos a utilizar en memoria, asignando además un valor a los datos para su posterior procesamiento.

Posteriormente, se recorrerá todo el volumen de datos realizando las operaciones de lectura o escritura en memoria, y por consiguiente, introduciendo carga entre la CPU y la memoria del sistema.

Una vez realizadas todas las operaciones sobre el volumen de datos, se liberará dicho espacio y se dará un descanso al procesador de dos segundos para volver a iniciar el algoritmo.

3.2 – ENTORNO DE DESARROLLO

En esta parte del documento se hará una breve descripción de todo el entorno tecnológico a nivel software que se ha utilizado, así como las ventajas que ofrecen con respecto a otros posibles productos o herramientas.

3.2.1 – LENGUAJE DE PROGRAMACIÓN: C

El lenguaje de programación C fue creado en 1972 por Dennis M. Ritchie en los laboratorios Bell. Se trata de un lenguaje orientado a la implementación de SSOO, concretamente Unix. Este SSOO también fue desarrollado en 1969 por los mismos laboratorios y con Dennis Ritchie entre el equipo de desarrolladores.

Se trata de un lenguaje fuertemente tipificado, que genera un código muy eficiente. Además, es el lenguaje más utilizado para crear software de sistemas, aunque también puede utilizarse para el desarrollo de aplicaciones.

Aparte de disponer de las estructuras típicas de los lenguajes de alto nivel, es posible utilizar construcciones que permiten un control a muy bajo nivel. Los compiladores, además ofrecen muchas extensiones a este lenguaje, entre ellas la de acceder directamente a memoria o dispositivos periféricos.

Gracias a que existen compiladores para casi todos los sistemas conocidos, se trata del lenguaje más portado en existencia.

Sus características principales son:

- Lenguaje simple, con funciones añadidas proporcionadas por bibliotecas.
- Permite programar con múltiples estilos debido a su flexibilidad.
- Utiliza un lenguaje de pre-procesado para tareas como la inclusión de múltiples archivos de código fuente.
- Acceso a memoria a bajo nivel mediante el uso de punteros
- Reducido conjunto de palabras clave.
- Paso de parámetros POR VALOR, pudiendo realizar el paso por referencia pasando a las funciones la dirección de memoria de los parámetros.
- Punteros a funciones y variables estáticas.
- Uso de estructuras que permiten tener datos agrupados que se manipulan como un todo.

Como principales carencias de este lenguaje de programación cabe citar que no tiene un recolector de basura nativo, así como soporte nativo para la programación multihilo y orientada a objetos.

Este lenguaje ha sido elegido para la realización del proyecto debido a las muchas ventajas que nos ofrece para la codificación de la aplicación a desarrollar en este proyecto.

3.2.2 – SISTEMAS OPERATIVOS

3.2.2.1 – WINDOWS 7

Versión más reciente de Microsoft Windows, una línea de sistemas operativos producidos por Microsoft. Diseñado para su uso de PC, especialmente en ordenadores de escritorio o portátiles.

Ilustración 4: Logotipo de Windows 7

Se trata de una versión incremental de su predecesor, Windows Vista, con un núcleo basado en NT 6.0, lo que le permite mantener un alto grado de compatibilidad con aplicaciones de las que éste era ya compatible.

Pero además de esto, su principales meta fue la mejora de su interfaz para hacerlo más accesible a los usuarios.

Junto a esto, se incluyen nuevas características que permiten la realización de tareas de forma más rápida y sencilla, mientras que se intenta lograr un sistema más estable y ligero.

Sobre este sistema operativo se realiza la mayor parte de la desarrollo del proyecto, ya que su entorno gráfico ofrece muchas ventajas a la hora de utilizar las diferentes herramientas.

3.2.2.2 – LINUX / UBUNTU

Ubuntu es un sistema operativo mantenido por Canonical Ltd y la comunidad de desarrolladores. Está basado en Debian [4]y utiliza un núcleo Linux, el principal ejemplo del software libre y basado en Unix.

Ubuntu está orientado al usuario nobel por su facilidad de uso. Abarca aproximadamente la mitad de las distribuciones Linux, con una tendencia ascendente entre los servidores web.

Ilustración 5: Logotipo de Ubuntu

En su última versión, soporta oficialmente las arquitecturas de 32 y 64 bits en ordenadores personales y servidores, y al igual que en la mayoría de distribuciones Linux, es capaz de actualizar todas las aplicaciones instaladas al mismo tiempo a través de repositorios.

Por último, cabe destacar que los usuarios pueden participar en el desarrollo de este sistema operativo, solucionando *bugs*, escribiendo código o probando versiones inestables del sistema.

Ilustración 6: Logotipo de Linux

Éste ha sido el sistema operativo utilizado para la realización de las pruebas unitarias de la aplicación.

[4] Comunidad de desarrolladores y usuarios que mantiene el sistema operativo GNU basado en software libre.

3.2.3 – MÁQUINA VIRTUAL: VMWARE PLAYER

Se trata de un sistema de virtualización por software, que proporciona un ambiente de ejecución similar a un computador físico. Permite simular varios computadores dentro de un mismo hardware de forma simultánea, permitiendo un mayor aprovechamiento de los recursos.

Al tratarse de una capa intermedia entre el software emulado (el SSOO) y el hardware, la velocidad de ejecución puede verse afectada, siendo suficiente en la mayoría de los casos.

Al contrario que VirtualPC[5], otra herramienta de virtualización que emula la plataforma, VMware la virtualiza, y las instrucciones se realizan directamente sobre el hardware físico, en vez de tener que hacer llamadas al sistema operativo principal.

Para la realización del proyecto se ha utilizado esta herramienta para virtualizar el sistema operativo Ubuntu, teniendo Windows 7 como sistema operativo principal.

3.2.4 – COMPILADORES Y OTRAS LIBRERÍAS

3.2.4.1 – COMPILADOR GCC

Compilador integrado del proyecto GNU, válido para multitud de lenguajes de programación, capaz de recibir un programa fuente y generar un ejecutable binario en el lenguaje de la máquina en la que se ejecutará.

Las siglas GCC significan “GNU Compiler Collection”, que originalmente significaban “GNU C Compiler”.

En el proceso de compilación aparecen cuatro etapas sucesivas: Las de pre-procesamiento, compilación, ensamblado y enlazado. Los comandos gcc y g++ son capaces de realizar todo este proceso de una sola vez.

[5] Propiedad de Microsoft.

3.2.4.2 – MPICH2

MPICH2 es una distribución portable de MPI (Interfaz de Paso de Mensajes) de alto rendimiento utilizada en computación paralela para aplicaciones de memoria distribuida.

La implementación original de MPICH (MPICH1) implementa el estándar MPI-1.1 mientras que la última implementación (MPICH2) implementa el estándar MPI-2.2.

Se trata de una implementación distribuida como *open-source*, probada en multitud de plataformas, incluidas Linux, Mac OS/X, Solaris y Windows.

MPICH2 es una de las implementaciones más populares de MPI, utilizada como base para la gran mayoría de implementaciones de MPI como IBM MPI, Intel MPI, Microsoft MPI, etc.

Su principal objetivo es proporcionar una implementación para importantes plataformas, incluyendo clusters, SMPs y procesadores paralelos masivos. También proporciona un vehículo para la investigación y el desarrollo de nuevos y mejores entornos de programación paralela.

Esta distribución de MPI será la utilizada para la compilación y ejecución de la aplicación desarrollada, ya que es la implementación presente en el clúster donde se realizarán las diferentes evaluaciones.

3.2.4.3 – POSIX THREADS (PTHREADS)

La librería de Pthreads cumple con los estándares POSIX y permite trabajar con distintos hilos de ejecución al mismo tiempo.

La principal diferencia entre un thread y un proceso es que los procesos no comparten memoria entre sí, mientras que los threads sí lo hacen. Para la creación de los threads se utilizan las funciones de la librería pthread, mientras que para la creación de procesos se utiliza la llamada al sistema fork(), que se encuentra accesible en todos los sistemas Unix.

Esta librería ha sido utilizada en el proyecto para la creación de diferentes threads en la aplicación Generador_Carga, en los que se ejecutarán los diferentes módulos de la misma.

3.2.5 – EDITORES DE TEXTO

3.2.5.1 – NOTEPAD++

Notepad++ es un editor de texto y de código fuente disponible para Microsoft Windows, con soporte para varios lenguajes de programación. Tiene muchas similitudes con el Bloc de Notas, ya que es muy simple, pero incluye opciones avanzadas para desarrolladores y programadores. Sus principales características son:

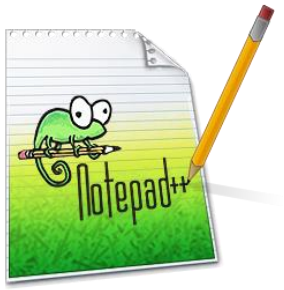


Ilustración 7: Logotipo de Notepad++

- Coloreado y envoltura de sintaxis: Es capaz de resaltar las expresiones propias de del lenguaje de programación en el que se está escribiendo.
- Permite abrir varios documentos en pestañas diferentes.
- Resaltado de apertura y cierre de paréntesis, llaves o corchetes.
- Soporte de extensiones

Notepad++ ha sido la herramienta utilizada en el proyecto para el desarrollo del código fuente de la aplicación.

3.2.5.2 – MICROSOFT WORD 2007

Software propiedad de Microsoft, creado para el procesamiento de textos y que viene integrado en la *suite* ofimática Microsoft Office 2007. Se trata del procesador de textos más popular del mundo y también está orientado al consumidor, por lo que no es gratuito.

Ilustración 8: Logotipo de Microsoft Word 2007

La versión actual para Windows es *Microsoft Office Word 2010*, aunque para la realización de la documentación del proyecto se ha utilizado la versión 2007, registrado mediante una licencia incluida en la compra del equipo en el que se ha realizado el desarrollo.

3.3 – ANÁLISIS DE REQUISITOS

En este apartado se realizará un análisis de aquellos requisitos que debe cumplir el software que se desarrolla en este proyecto. Los requisitos están definidos a partir de las especificaciones dadas por el tutor del proyecto, y el desarrollador del mismo, teniendo en cuenta las diferentes opciones y restricciones que se dan en todo el ciclo de vida del software.

Los requisitos se han definido mediante reuniones con el tutor, con el objetivo de plasmar de forma clara las necesidades que la aplicación debe de cubrir. Además, como ya se ha comentado, el desarrollador del proyecto ha añadido algún nuevo requisito durante la fase de desarrollo del mismo.

3.3.1 – RESTRICCIONES

En el desarrollo de la aplicación hay una serie de restricciones que hacen que el desarrollo no sea óptimo. Entre estas restricciones encontramos las siguientes:

- *Restricciones de carácter económico*: Al tratarse de un proyecto académico, de Fin de Grado, el alumno debe afrontar con todo el trabajo y los costes del desarrollo del mismo. Debido a esto, se han utilizado herramientas gratuitas para la realización de todas las fases de desarrollo del software, desde el análisis hasta la documentación final del mismo.
- *Restricciones de carácter técnico*: La modificación de los parámetros de los diferentes módulos debe hacerse de forma manual, editando el código fuente de la aplicación, lo que obliga al usuario a prestar especial atención a la guía del usuario que aparece como anexo al presente documento.

3.3.2 – IDENTIFICACIÓN DE REQUISITOS DE USUARIO

Para la identificación de los requisitos se ha seguido la técnica de Métrica V3, introduciendo aquellos atributos que se han considerado más importantes para la interpretación de los requisitos.

A la hora de nombrar los requisitos, utilizaremos la siguiente notación:

UR-[C | R]nnn

Donde el significado de sus componentes es:

- UR: Requisito de usuario (User requeriment).
- C: Requisito de capacidad.
- R: Requisito de restricción.
- nnn: Número identificativo del requisito.

En cuanto a los atributos de cada requisito, indicaremos los siguientes:

- *Descripción*: Breve comentario sobre las necesidades que el requisito expone.
- *Fuente*: Indica el origen del requisito. El requisito podrá ser propuesto por el tutor o por el desarrollador.
- *Necesidad*: Indica el nivel de exigencia con el que el requisito debe de cumplirse en el software. Podrán incluirse requisitos con necesidad **Esencial**, cuyo cumplimiento es obligatorio, **Deseable** o **Opcional**, estos dos últimos siendo de menor importancia, respectivamente.
- *Prioridad*: Indica el orden de implementación del requisito. La prioridad de un requisito podrá ser **Alta**, **Media** o **Baja**.
- *Estabilidad*: Define el nivel de invariabilidad del requisito a lo largo del ciclo de vida del software. Al igual que la prioridad, podrá tomar los valores **Alta**, **Media** o **Baja**

Una plantilla para la definición de requisitos será la siguiente:

Identificador			
Descripción			
Fuente	☐ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☐ Deseable	☐ Opcional
Prioridad	☐ Alta	☐ Media	☐ Baja
Estabilidad	☐ Alta	☐ Media	☐ Baja

Tabla 2: Plantilla de definición de Requisitos

3.3.3 – REQUISITOS DE CAPACIDAD

Los requisitos de capacidad definen la función general del requisito. Los requisitos de capacidad que se han definido para la realización del software final son los siguientes:

UR-C001			
Descripción	La aplicación parará su ejecución solamente al ser abortada por el usuario.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☐ Alta	☒ Media	☐ Baja

Tabla 3: Requisito de Capacidad UR-C001

UR-C002			
Descripción	El programa de evaluación podrá lanzar varios procesos de forma simultánea.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 4: Requisito de Capacidad UR-C002

UR-C003			
Descripción	El programa de evaluación podrá contener tres módulos diferenciados: Un módulo de comunicaciones de red, un módulo de CPU y un módulo de memoria.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☐ Alta	☒ Media	☐ Baja

Tabla 5: Requisito de Capacidad UR-C003

UR-C004			
Descripción	La ejecución de los diferentes módulos deberá ser concurrente.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 6: Requisito de Capacidad UR-C004

UR-C005			
Descripción	Cada uno de los módulos podrá separarse del programa principal para ejecutarse en solitario.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Estabilidad	☐ Alta	☐ Media	☒ Baja

Tabla 7: Requisito de Capacidad UR-C005

UR-C006			
Descripción	En el módulo de comunicaciones de red se podrán definir diferentes tipos de comunicaciones.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 8: Requisito de Capacidad UR-C006

UR-C007			
Descripción	Las comunicaciones se harán entre pares de procesos.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 9: Requisito de Capacidad UR-C007

UR-C008			
Descripción	Las comunicaciones entre procesos estarán definidas en un archivo de texto externo al programa.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 10: Requisito de Capacidad UR-C008

UR-C009			
Descripción	El módulo de CPU introducirá cargas computacionales al procesador de forma controlada, manteniendo el límite de utilización especificado.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 11: Requisito de Capacidad UR-C009

UR-C010			
Descripción	El módulo de CPU variará su comportamiento dependiendo de las condiciones locales del sistema o nodo.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 12: Requisito de Capacidad UR-C010

UR-C011			
Descripción	El módulo de Memoria aumentará el tráfico entre el procesador y la memoria de forma controlada.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 13: Requisito de Capacidad UR-C011

UR-C012			
Descripción	El porcentaje de lecturas o escrituras del módulo de Memoria podrá ser definido por el usuario.		
Fuente	☐ Tutor	☒ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 14: Requisito de Capacidad UR-C012

UR-C013			
Descripción	El módulo de memoria calculará el número de bloques accedidos de memoria, así como el tiempo necesario para su realización atendiendo a los datos de una arquitectura dada.		
Fuente	☐ Tutor	☒ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 15: Requisito de Capacidad UR-C013

3.3.4 – Requisitos de Restricción

Los requisitos de restricción definen la forma en la que se realizarán los requisitos de capacidad definidos anteriormente. Se puede decir que restringen la forma en la que se resuelve el requisito de usuario.

Los requisitos de restricción que se han definido son los siguientes:

UR-R001			
Descripción	El programa de evaluación lanzará los procesos como procesos MPI		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 16: Requisito de Restriccion UR-R001

UR-R002			
Descripción	El programa de evaluación deberá estar escrito en el lenguaje de programación C.		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 17: Requisito de Restriccion UR-R002

UR-R003			
Descripción	Los diferentes módulos del programa de evaluación se lanzarán como hilos de ejecución o “threads” utilizando la librería Pthread.		
Fuente	☒ Tutor	☒ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 18: Requisito de Restriccion UR-R003

UR-R004			
Descripción	Los diferentes tipos de comunicaciones que se utilizarán en el módulo de comunicaciones se definirán a partir de 3 parámetros: • Tamaño del mensaje • Tiempo de espera para iniciar comunicación • Proceso emisor del mensaje		
Fuente	☒ Tutor	☐ Desarrollador	
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 19: Requisito de Restriccion UR-R004

UR-R005			
Descripción	Las comunicaciones entre los diferentes pares de procesos se harán a través de las llamadas punto a punto de MPI (MPI_Send y MPI_Recv).		
Fuente	☒ Tutor	☒ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☒ Alta	☐ Media	☐ Baja

Tabla 20: Requisito de Restriccion UR-R005

UR-R006			
Descripción	La definición de los tipos de comunicación entre procesos se hará a través de una matriz codificada en un archivo de texto plano.		
Fuente	☐ Tutor	☒ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Estabilidad	☐ Alta	☒ Media	☐ Baja

Tabla 21: Requisito de Restriccion UR-R006

UR-R007			
Descripción	El módulo de CPU comprobará las condiciones locales del sistema a través de llamadas al sistema que permitan conocer el porcentaje de utilización de CPU del hilo.		
Fuente	☒ Tutor	☒ Desarrollador	
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Estabilidad	☐ Alta	☒ Media	☐ Baja

Tabla 22: Requisito de Restriccion UR-R007

4 – GESTIÓN DEL PROYECTO

4.1 - PREDICCIÓN DE COSTES

Para la realización de este proyecto se ha descrito una predicción de recursos tanto software como hardware, tiempo, costes y esfuerzo necesarios para el desarrollo del mismo.

Utilizaremos el modelo de estimación denominado **COCOMO II**, que permite realizar estimaciones a partir del tamaño del código fuente y un conjunto de factores de escala y de costo. La herramienta dispone de dos modelos diferenciados como son: *Diseño Preliminar* y *Post-Arquitectura*.

Para llevar a cabo la estimación en base al modelo anteriormente citado, se ha utilizado la herramienta USC-COCOMO II.2000.0. Esta herramienta requiere la introducción de una serie de parámetros para realizar la estimación. Estos datos son los siguientes:

- *Nombre del Proyecto (Project Name)*: Nombre del proyecto. Para este proyecto el nombre será "TFG-jfcanseco".
- *Nombre del Módulo (Module Name)*: Utilizado para diferenciar diferentes módulos dentro de un proyecto. Para realizar la estimación de este proyecto se definirá un único módulo. El nombre del módulo definido es "TFG-Modulo1"
- *Tamaño del módulo (Module Size)*: Representa el tamaño en líneas de código del módulo. El valor se puede representar de varias maneras:
    1. Introduciendo el número de líneas de código en el campo SLOC.
    2. Utilizando el modelo de los puntos de función.
    3. Utilizando el *"Adaptation Adjustement Factor"* (Factor de ajuste de adaptación).

  Todas las opciones permiten además de indicar el lenguaje de programación en el que se realiza el desarrollo, indicar un parámetro *REVL*, en el que se indica el porcentaje de código que será descartado por la volatilidad de los requisitos. Para realizar la estimación del proyecto se utiliza la primera opción, y se introducen las líneas de código implementadas.

- *Sueldo Mensual (LABOR Rate):* Este campo contiene la cantidad de dinero que percibe al mes el equipo de desarrollo de cada módulo por separado. Fijando un salario de 25 €/hora para el desarrollador, y fijando el número de días laborables de un mes a 21, invirtiendo 4 horas en cada día de trabajo, se estima un coste mensual de 2100 €/mes.
- *Factor de ajuste del esfuerzo (EAF):* Representa una serie de multiplicadores de esfuerzo. El número de parámetros varía según el modelo que esté seleccionado (*Diseño Preliminar* o *Post-Arquitectura*).
- *Factor de escala (Scale Factor):* Parámetros globales del proyecto.
- *Riesgo (RISK):* Nivel de riesgo para cada módulo. Solamente se aplica cuando se utiliza el modelo *Post-Arquitectura*

4.1.1 – Estimación Inicial

Para la estimación de los distintos factores de esfuerzo se introducen los siguientes multiplicadores:

EAF	
RCPX: *Fiabilidad del software*	ALTA
RUSE: *Reutilización de código*	BAJA
PDIF: *Dificultad de la plataforma*	BAJA
PERS: *Capacidad del personal*	ALTA
PREX: *Experiencia del personal*	ALTA
FCIL: *Facilidades*	ALTA

Tabla 23: Estimación inicial: Factores de esfuerzo

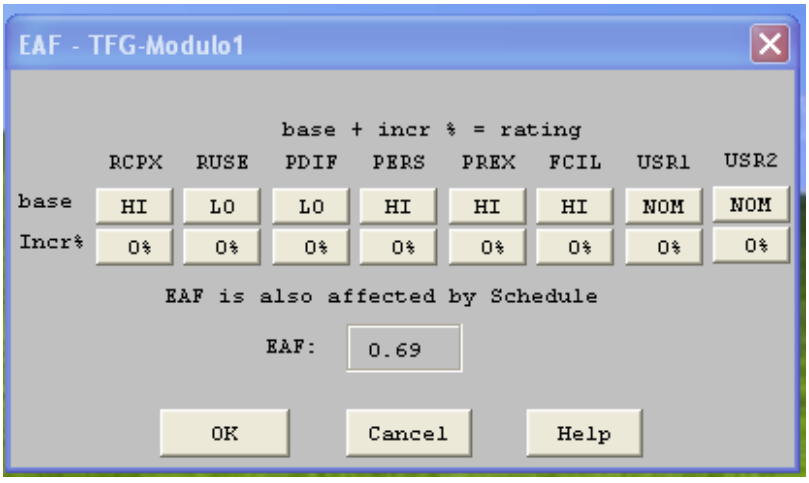


Ilustración 9: Estimación inicial: Factores de esfuerzo

En segundo lugar, introduciremos los valores de los factores de escala:

Scale Factor	
Precedentedness: *Experiencia*	ALTA
Developement Flexibility: *Flexibilidad en el desarrollo*	BAJA
Architecture/Risk resolution: *Riesgo*	NOMINAL
Team Cohesion: *Unión del Grupo*	MUY ALTA
Process Madurity: *Nivel de madurez*	NOMINAL

Tabla 24: Estimación inicial: Factores de escala

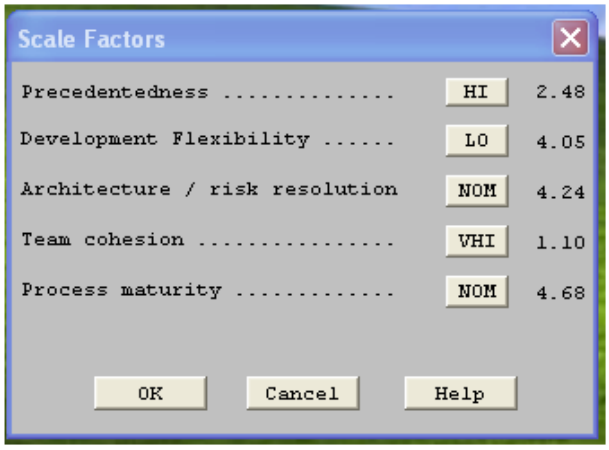


Ilustración 10: Estimación inicial: Factores de escala

Con estas estimaciones, COCOMO nos realiza los siguientes cálculos:

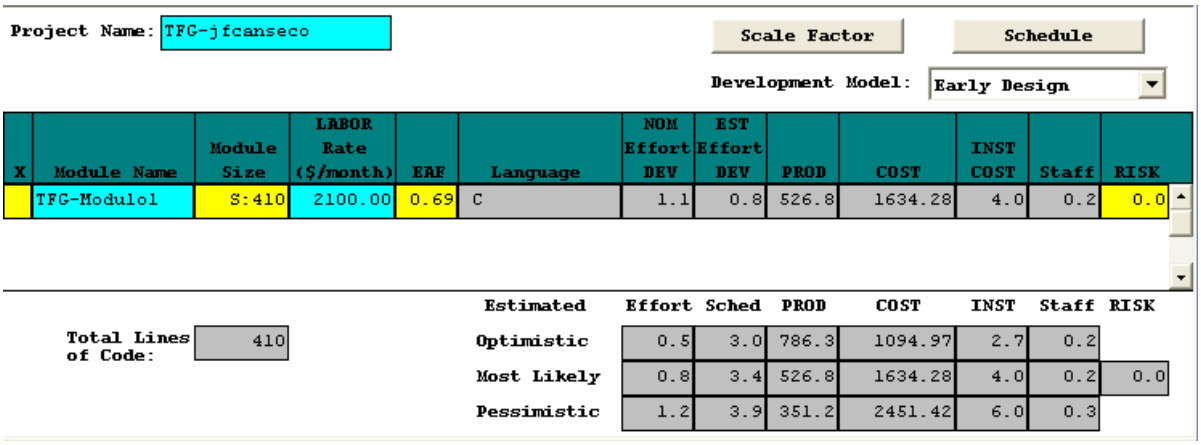


Ilustración 11: Estimación inicial de COCOMO II

4.1.2 – ESTIMACIÓN FINAL

Mientras los factores de escala se mantienen, para la estimación final introducimos los siguientes parámetros en la tabla de factores de esfuerzo:

EAF	
RELY: ***Fiabilidad***	ALTA
DATA: ***Tamaño datos aplicación***	BAJO
DOCU: ***Concordancia del desarrollo de la documentación con el ciclo de vida del software***	NOMINAL
CPLX: ***Complejidad***	NOMINAL
RUSE: ***Reutilización***	BAJO
TIME: ***Uso de tiempo de ejecución***	ALTO
STOR: ***Uso de memoria principal***	ALTO
PVOL: ***Volatilidad del código***	ALTA
ACAP: ***Capacidad del analista***	ALTA
PCAP: ***Capacidad del programador***	ALTA
PCON : ***Continuidad del personal***	MUY ALTA
APEX: ***Experiencia en aplicaciones***	ALTA
LTEX: ***Experiencia con lenguaje y herramientas***	MUY ALTA
PLEX: ***Experiencia con la plataforma***	BAJA
TOOL: ***Uso de herramientas de software***	ALTA
SITE: ***Diferentes lugares de desarrollo***	NOMINAL

Tabla 25: Estimación final: Factores de esfuerzo

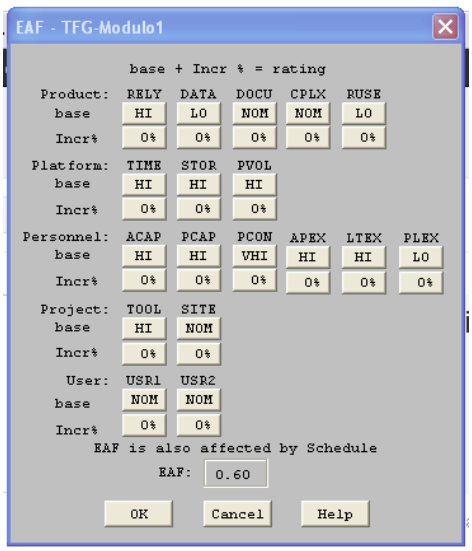
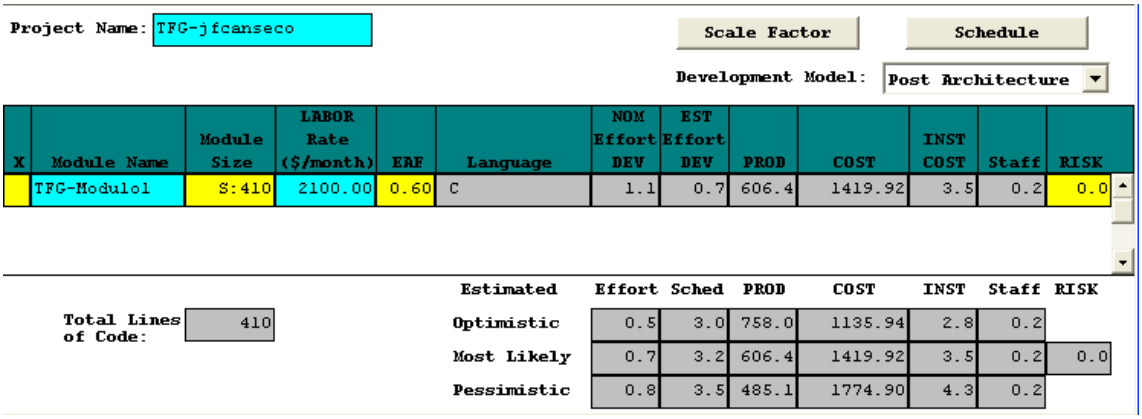


Ilustración 12: Estimación final: Factores de esfuerzo

La estimación final generada por la herramienta es:



Project Name: TFG-jfcanseco

Scale Factor | Schedule

Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	TFG-Modulo1	S:410	2100.00	0.60	C	1.1	0.7	606.4	1419.92	3.5	0.2	0.0

Total Lines of Code: 410

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.5	3.0	758.0	1135.94	2.8	0.2	
Most Likely	0.7	3.2	606.4	1419.92	3.5	0.2	0.0
Pessimistic	0.8	3.5	485.1	1774.90	4.3	0.2	

Ilustración 13: Estimación final con COCOMO II

4.1.3 – CONCLUSIONES

Tanto en la estimación inicial como en la final obtenemos unos valores bastante parecidos, solamente notando diferencia en que la estimación del coste del proyecto es algo menor al hacer la estimación final. Esta diferencia puede deberse a que en el diseño previo no se introducen muchos factores de estimación que en el diseño posterior sí que son conocidos. El coste que obteníamos en la estimación inicial era de 1634.28 €, mientras que el de la final es de 1419,92 €.

En cuanto al número de personas necesarias para la implementación, en ambas estimaciones obtenemos el mismo valor, de una sola persona (0.8), ya que el desarrollo, al disponer de pocas líneas de código se estima que se realizará en poco menos de un mes.

Hay una gran diferencia entre el coste estimado por COCOMO II al obtenido en el apartado 7.4 – Coste Total, en el que se da el coste total presupuestado del proyecto, cuyo coste es de **15.571 €**. Esta diferencia puede deberse a la forma en la que la herramienta COCOMO II estima el esfuerzo, ya que estima por sí misma la dedicación de horas, siendo el coste mensual introducido más bajo debido a que el número de horas trabajadas por jornada es de cuatro.

4.2 – PLANIFICACIÓN TEMPORAL

Para realizar la planificación temporal del proyecto ha sido necesario determinar en primer lugar las fases y el orden en el que se va a dividir el desarrollo del software.

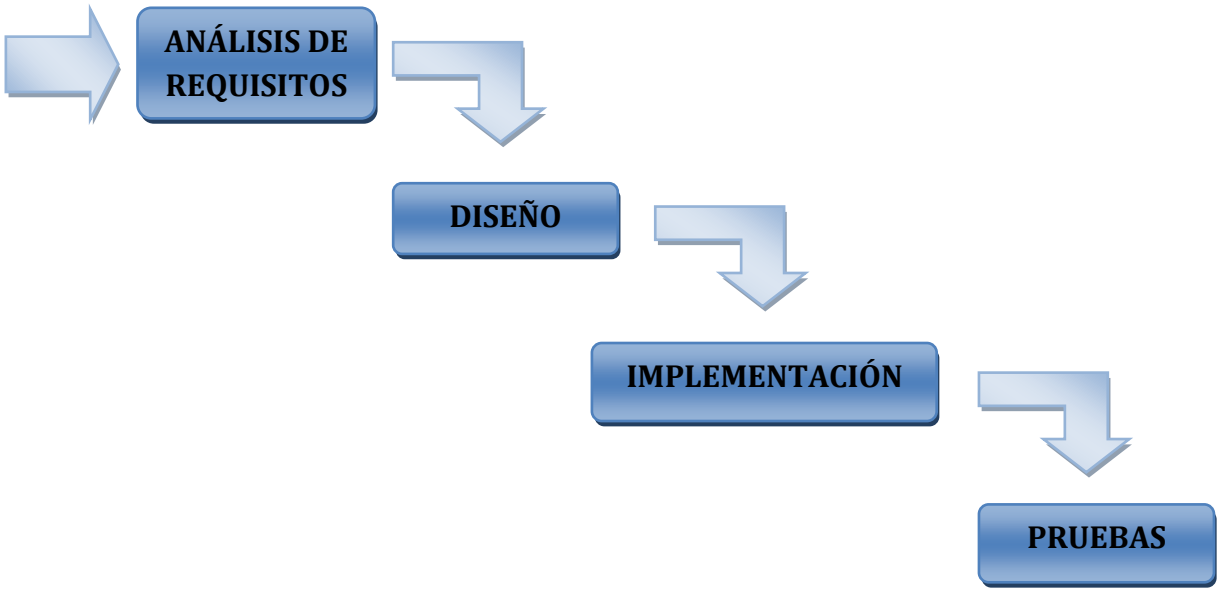


Ilustración 14: Ciclo de vida del software

Como podemos ver en la ilustración, se ha elegido una gestión del software en cascada, en la que se van realizando diferentes fases. Según este modelo, todos aquellos cambios realizados en una fase del desarrollo repercutirán en las fases anteriores y posteriores.

Durante la fase de implementación se podrán realizar pruebas unitarias de forma paralela para verificar la correcta implementación de los diferentes módulos por separado. La última fase de pruebas se refiere a las pruebas de la aplicación completa, estando ya terminada la implementación de la misma.

La fecha de inicio del proyecto es el día **1 de Febrero de 2012**, y se estima que la fecha de finalización será el **5 de Septiembre de 2012**. Suponiendo que se dedican 5 días a la semana a la realización del proyecto, la suma total de días de trabajo es de 155 días.

Las diferentes fases que se han definido para el desarrollo completo del proyecto son las siguientes:

- **Inicio**
  - Problemática y definición de objetivos
- **Estado de la cuestión**
  - Estudio de las diferentes tecnologías
- **Gestión del proyecto**
  - Metodología a emplear
  - Organización del trabajo
- **Análisis**
  - Definición de requisitos de usuario
- **Diseño**
  - Descripción del entorno de desarrollo
  - Descripción de la arquitectura a nivel software
- **Implementación**
  - Codificación del software
  - Pruebas unitarias
- **Pruebas**
  - Pruebas en entorno real
- **Documentación**
  - Realización de la memoria
  - Generación de manuales de usuario

Una vez definida la planificación llevada a cabo para el desarrollo del proyecto se utilizará un diagrama de Gantt para ubicar todas las fases seguidas en el desarrollo del mismo, así como su ordenación temporal en el tiempo.

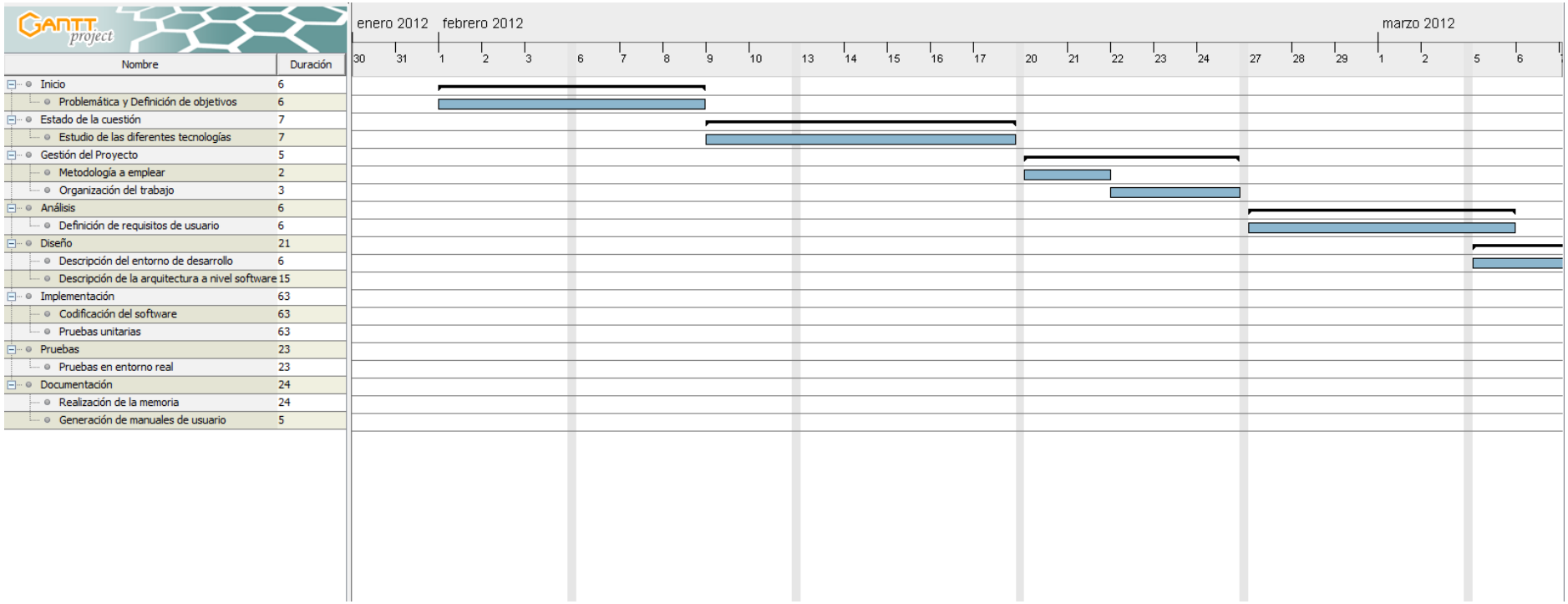


Ilustración 15: Diagrama de Gantt (1)

En esta primera figura podemos observar la distribución en el tiempo de las primeras fases del proyecto, desde la fecha de inicio del mismo, definida el día 1 de Febrero de 2012. La fase inicial tiene una duración de 6 días, en la que se define la problemática y la definición de objetivos. Para el estudio de las diferentes tecnologías se planifican un total de 7 días, entre los días 9 y 20 de febrero. En cuanto a la gestión del proyecto se planifican un total de 5 días, repartidos 2 y 3 días en las tareas de selección de la metodología a emplear y organización del trabajo, respectivamente. Por último, la fase de definición de requisitos tiene una duración de 6 días, ubicada entre el 27 de febrero y el 6 de marzo. De forma paralela a la fase de análisis, se comienza con la fase de diseño.

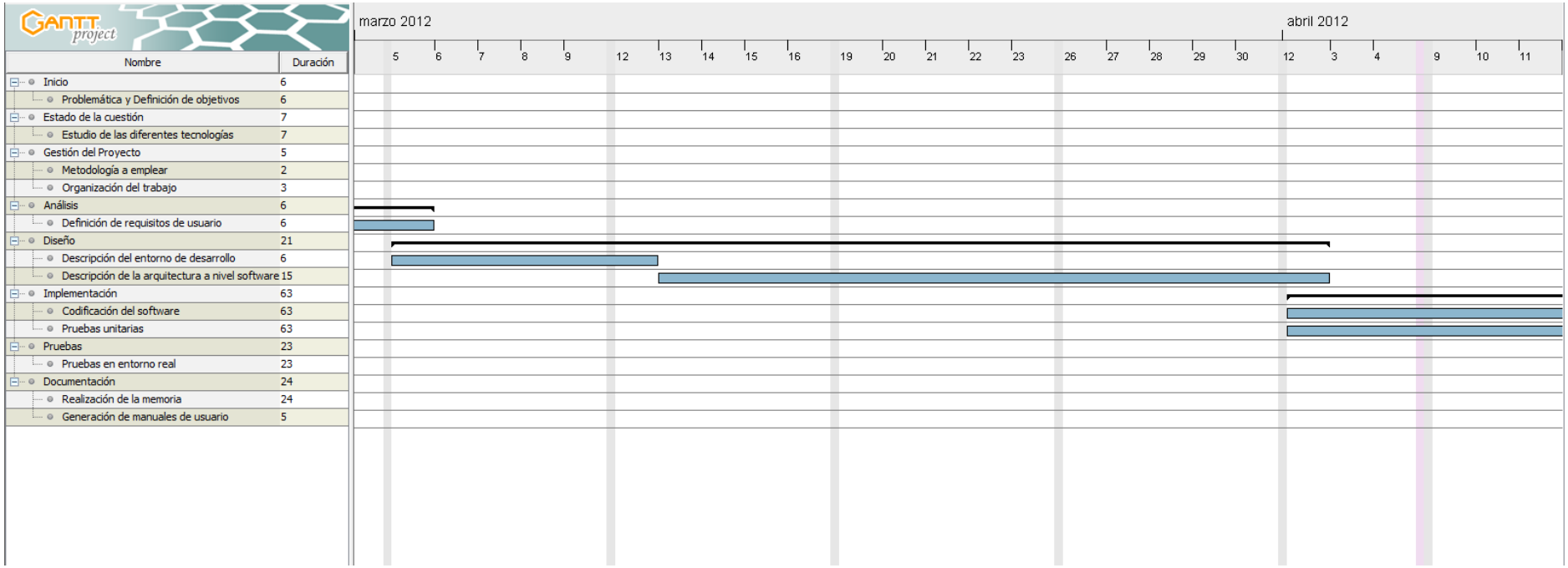


Ilustración 16: Diagrama de Gantt (2)

Como se puede apreciar en la ilustración, la primera subfase de la fase de diseño tiene una duración de 6 días (Descripción del entorno de desarrollo), mientras que la subfase de Descripción de la arquitectura a nivel software tiene una duración de 15 días. La suma de estas 2 subfases hace un total de 21 días para la fase de diseño. Al final de la figura se puede apreciar el comienzo de la fase de implementación, coincidiendo en el tiempo con los últimos días del diseño.

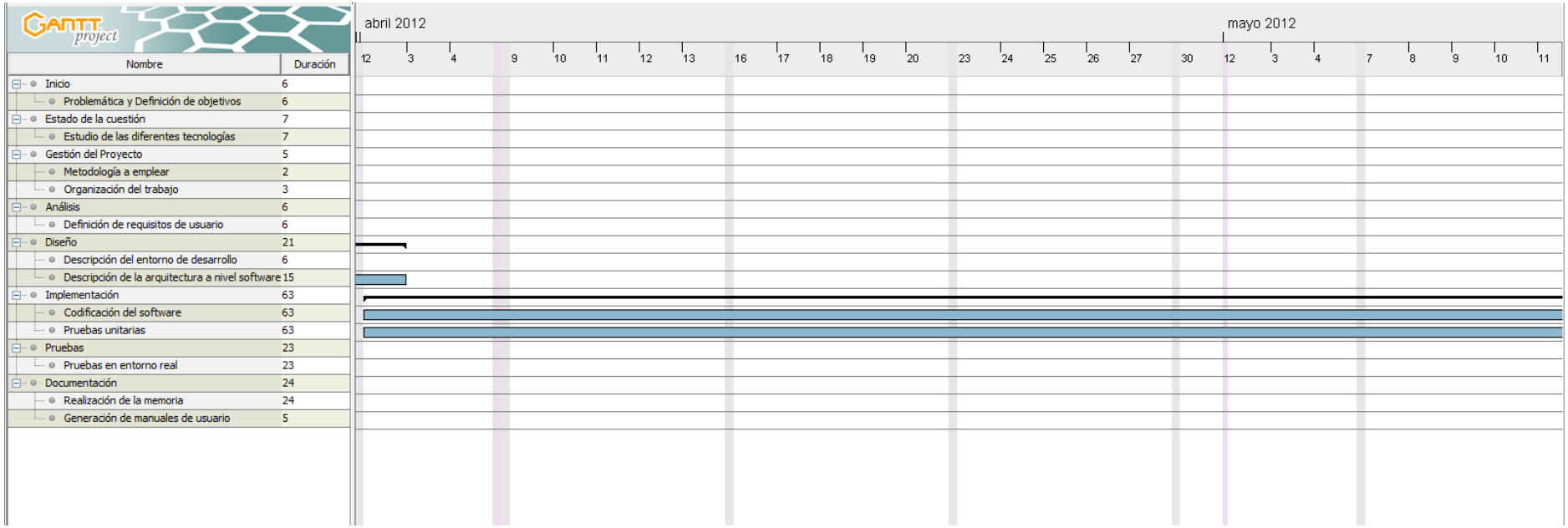


Ilustración 17: Diagrama de Gantt (3)

En la fase de implementación se puede observar que se realizará de forma paralela tanto la implementación del software como las pruebas unitarias del mismo, con el objetivo de poder corregir posibles errores de forma más eficiente y rápida. La duración de esta fase de desarrollo es de 63 días.

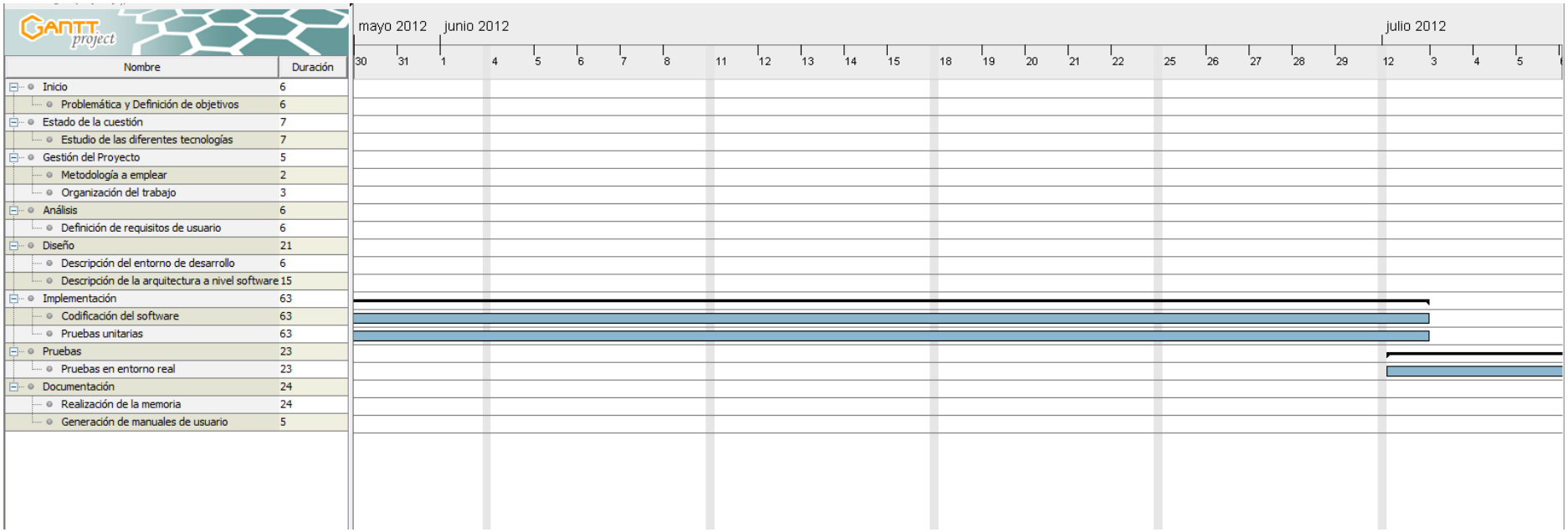


Ilustración 18: Diagrama de Gantt (4)

Al igual que en fases anteriores del desarrollo del proyecto, la fase de pruebas en entorno real o evaluación de la aplicación junto a aplicación paralela, se inicia sin haber terminado la fase de implementación. Al disponer de varios módulos dentro de la aplicación, los módulos ya implementados y probados pueden ser evaluados junto a la aplicación paralela en el entorno de pruebas definido para ello.

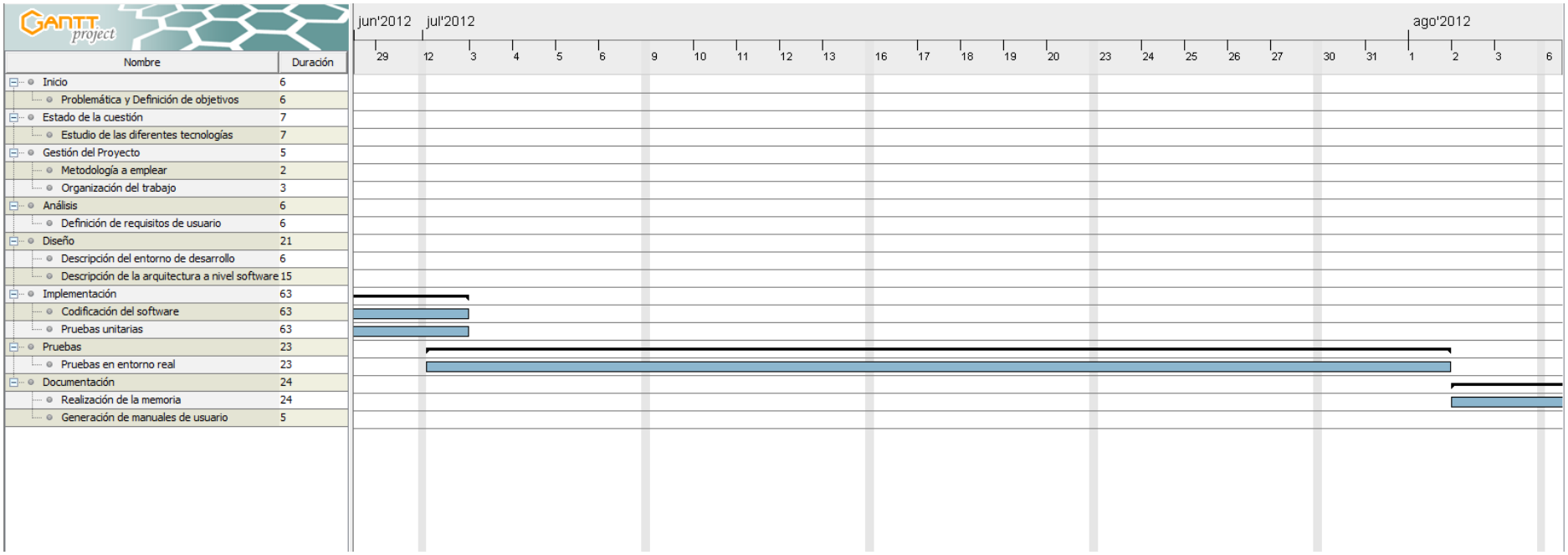


Ilustración 19: Diagrama de Gantt (5)

En esta ilustración observamos la finalización de la fase de pruebas, concluidos los 23 días planificados para su realización y el comienzo de la fase de documentación. Esta fase contiene la subfase de la realización de la memoria, cuya duración estimada es de 24 días y la de Generación de los manuales de usuario, con una duración estimada de 5 días. En total, la fase de Documentación tiene una duración de 29 días.

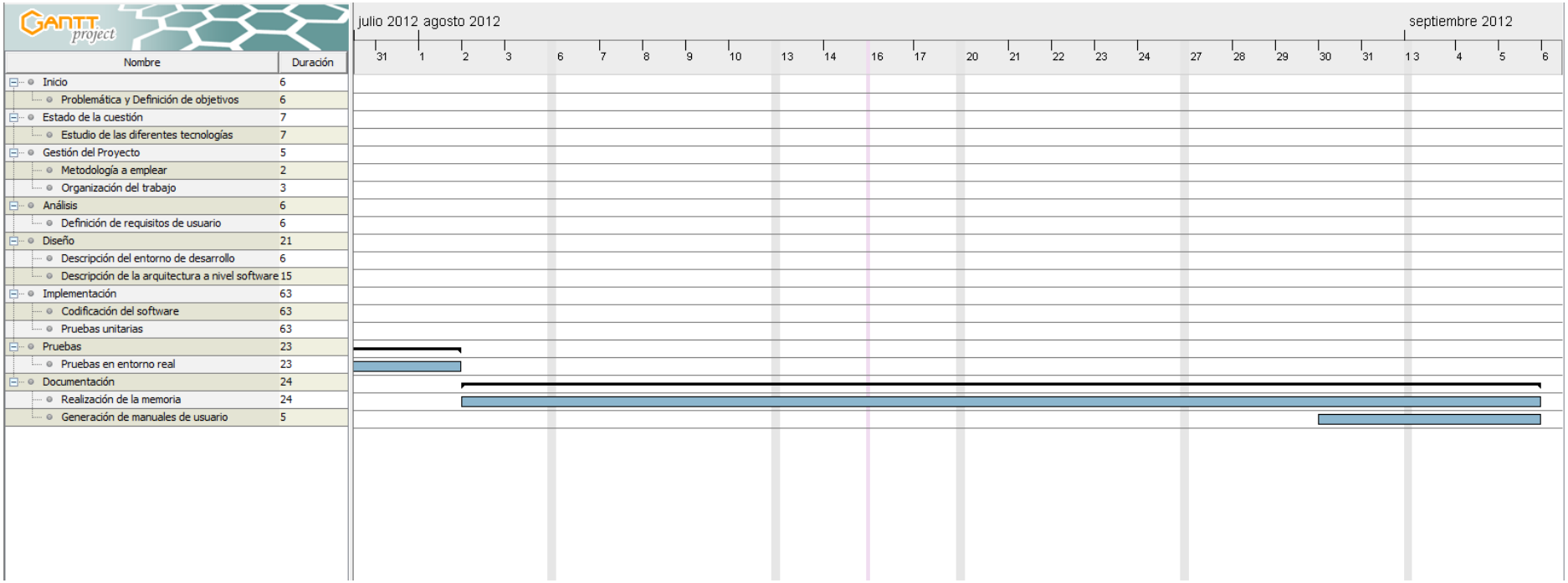


Ilustración 20: Diagrama de Gantt (6)

En la Ilustración 20: Diagrama de Gantt (6) se muestra el fin de la planificación del proyecto, establecido el día 5 de Septiembre de 2012, completando la suma de 155 días de trabajo estimados.

5 – PRUEBAS

En el presente apartado del documento se definirán las pruebas a realizar para comprobar que el comportamiento del software es el requerido, y que cumple las expectativas creadas.

Una vez confirmado el correcto funcionamiento del software, se procederá a la realización de distintas evaluaciones en un entorno real, realizando ejecuciones de la aplicación junto a un proceso paralelo, con el objetivo de medir los efectos que el software genera en el entorno de ejecución.

5.1 – DESCRIPCIÓN DEL ENTORNO DE PRUEBAS

Para la realización de las pruebas se han utilizado diferentes entornos:

5.1.1 – PRUEBAS ESPECÍFICAS DE LA APLICACIÓN

Para realizar las pruebas unitarias de la aplicación se han utilizado los siguientes equipos:

- Ordenador Portátil Packard Bell EasyNote LJ75:
  - *Procesador*: Intel Core i5 M430 2,27 GHz
  - *Memoria RAM*: 4 GB
  - *Disco duro*: Toshiba MK5076GSX 500 GB
  - *Sistema Operativo*: Ubuntu 12.04 (virtualizado sobre Windows 7 Professional Service Pack 1)

- Ordenador de sobremesa Packard Bell
  - *Procesador:* AMD Athlon 64 X2 Dual Core Processor 4200+ 2,19 GHz
  - *Memoria RAM*: 1 GB
  - *Disco Duro:* ST3400832AS 350 GB
  - *Sistema Operativo*: Ubuntu 12.04 (virtualizado sobre Windows XP Home Edition Service Pack 3)

Ambos equipos disponen, como hemos comentado en el apartado 3.2 – Entorno de desarrollo, de una máquina virtual en la que se ejecutará el sistema operativo Ubuntu.

Para la compilación de la aplicación y su posterior ejecución se utiliza la librería MPICH2, que implementa el estándar MPI-2.1.

5.1.2 – Evaluación junto a Aplicación Paralela

Para la realización de las pruebas de la aplicación junto a una aplicación paralela, el entorno de simulación se corresponde con el clúster *Tucan*, perteneciente al grupo de investigación de Arquitectura de Computadores, Comunicaciones y Sistemas (ARCOS), del Departamento de Informática de la Universidad Carlos III de Madrid.

El clúster *Tucan* está compuesto por seis grupos de equipos informáticos de diferente propósito y características, como por ejemplo computación GPGPU. El clúster está controlado por un PBS (Portable Batch System), que es un software de gestión de planificación de los trabajos lanzados sobre el clúster.

Para desarrollar la evaluación del proyecto se ha empleado una sección del clúster cuyos nodos cuentan con las siguientes características técnicas:

- *Número de nodos* : 24
- *Procesador*: Intel Xeon E5405
- *Velocidad del procesador*: 2.00 GHz
- *Número de procesadores por nodo*: 4
- *Memoria*: 4 Gigabytes
- *Disco duro*: 1 Terabyte
- *Red de datos*: InfiniBand
- *Sistema Operativo*: Linux Ubuntu 10.10
- *Kernel*: 2.6.35-28-server

Al igual que en el entorno de desarrollo y pruebas unitarias de la aplicación, para la compilación y la ejecución de los trabajos se utilizará la librería MPICH2, que implementa el estándar MPI-2.1.

Aquí podemos ver una representación de la arquitectura del clúster Tucan:

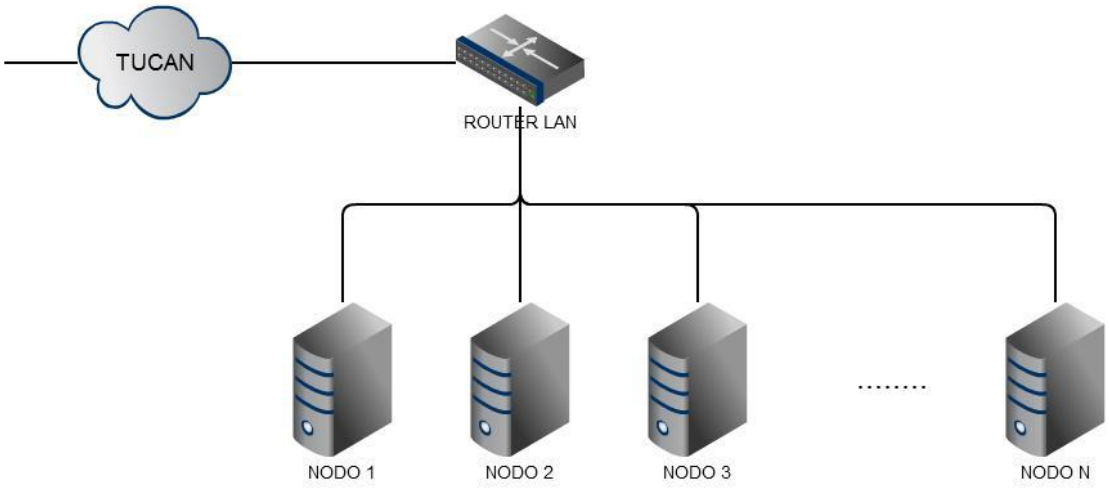


Ilustración 21: Arquitectura del clúster de pruebas

Como anteriormente se ha descrito, cada nodo del clúster tiene cuatro procesadores Intel Xeon E5405.

5.2 – PRUEBAS DE LA APLICACIÓN

En este apartado se definen las pruebas que se han de hacer para garantizar el cumplimiento de los requisitos recogidos por parte del tutor del proyecto, analizando posteriormente el resultado de las ejecuciones de los diferentes módulos de la aplicación.

Se definirán pruebas para los requisitos de capacidad del software, y no así para los de capacidad, ya que estos requisitos no especifican funciones que el software debe hacer, sino la forma específica en la que debe de hacerlo.

Finalmente, se añade una matriz de trazabilidad en la que se muestra la relación de las pruebas con los diferentes requisitos.

Las diferentes pruebas que se han realizado son las siguientes:

PRUE-1	
Objetivo de la prueba	Lanzamiento de el módulo de comunicaciones
Requisito/s relacionado/s	UR-C002,UR-C005,UR-C006,UR-C007,UR-C008
Condiciones de entrada	1. Ejecutar hilo de comunicaciones. 2. Ejecutar con la opción de MPI de generar un número de procesos superior a 1. 3. Lectura del archivo "matriz.txt" con la matriz de comunicaciones a realizar por los diferentes procesos. 4. Definidos 3 tipos diferentes de comunicaciones en la función.
Resultado	**ÉXITO**. Se realizan las comunicaciones entre los diferentes procesos de la forma esperada.

Tabla 26: Identificador de la Prueba (1)

PRUE-2	
Objetivo de la prueba	Lanzamiento de el módulo de CPU
Requisito/s relacionado/s	UR-C005,UR-C009,UR-C010
Condiciones de entrada	1. Ejecutar hilo de CPU. 2. Introducir un valor límite de porcentaje de utilización de la CPU.
Resultado	**ÉXITO**. Tras un tiempo de ajuste la aplicación se ajusta al porcentaje de uso del procesador, y reaccionando a los cambios existentes en el entorno.

Tabla 27: Identificador de la Prueba (2)

PRUE-3	
Objetivo de la prueba	Lanzamiento de el módulo de Memoria
Requisito/s relacionado/s	UR-C005,UR-C011,UR-C012,UR-C013
Condiciones de entrada	1. Ejecutar hilo de Memoria. 2. Introducir los datos propios de la arquitectura a analizar. 3. Introducir tamaño del volumen de datos y del ancho de banda. 4. Introducir porcentaje de lecturas/escrituras a realizar en memoria.
Resultado	**ÉXITO**. Se realiza el cálculo del número de bloques de memoria que serán accedidos, así como el tiempo necesario para realizar la operación. Además, se comprueba que se cumple el porcentaje introducido en las operaciones de lectura y escritura en memoria.

Tabla 28: Identificador de la Prueba (3)

PRUE-4	
Objetivo de la prueba	Lanzamiento conjunto de los diferentes módulos
Requisito/s relacionado/s	UR-C002,UR-C003,UR-C004
Condiciones de entrada	1. Condiciones de entrada de PRUE-1. 2. Condiciones de entrada de PRUE-2. 3. Condiciones de entrada de PRUE-3. 4. Habilitar la ejecución de los tres hilos.
Resultado	**ÉXITO**. Correcta ejecución de forma concurrente de los diferentes módulos en múltiples procesos.

Tabla 29: Identificador de la Prueba (4)

PRUE-5	
Objetivo de la prueba	Parada de la aplicación
Requisito/s relacionado/s	UR-C001
Condiciones de entrada	1. Condiciones de entrada de PRUE-4
Resultado	**ÉXITO**. El software termina su ejecución únicamente si el usuario termina la ejecución del proceso.

Tabla 30: Identificador de la Prueba (5)

Una vez definidas las pruebas, podemos hacer una relación entre las diferentes pruebas y los requisitos de usuario generando la siguiente ***matriz de trazabilidad:***

	UR-C001	UR-C002	UR-C003	UR-C004	UR-C005	UR-C006	UR-C007	UR-C008	UR-C009	UR-C010	UR-C011	UR-C012	UR-C013
PRUE-1		X			X	X	X	X					
PRUE-2					X				X	X			
PRUE-3					X						X	X	X
PRUE-4		X	X	X									
PRUE-5	X												

Tabla 31: Matriz de trazabilidad

*En el ANEXO 2: Ejemplos de Ejecución de la Aplicación, podrán visualizarse los resultados generados por algunas ejecuciones de ejemplo de los diferentes módulos, y poder así comprobar los resultados de las pruebas realizadas.

6 – EVALUACIÓN JUNTO A APLICACIÓN PARALELA EN ENTORNO DISTRIBUIDO

Una vez probado el correcto funcionamiento de la aplicación, y demostrado el cumplimiento de los requisitos, se procede a realizar una evaluación del impacto que tiene la aplicación en un entorno real.

Una vez que el entorno utilizado para la realización de esta evaluación ha sido descrito en el punto anterior, se da una pequeña descripción de la aplicación paralela con la que se realiza el estudio.

6.1 – APLICACIÓN PARALELA

La aplicación utilizada para realizar la evaluación, denominada *Gradiente,* será al igual que la aplicación *Generador_Cargas* desarrollada para el proyecto, una aplicación MPI.

La aplicación consiste en la resolución de un sistema de ecuaciones lineales con matrices, de forma iterativa. El número de iteraciones necesarias para la obtención del resultado se define como parámetro de entrada a la función.

La aplicación MPI puede ejecutarse de forma paralela, utilizando diferentes procesos, que haciendo uso de las llamadas colectivas del estándar MPI, permiten reducir el tiempo de ejecución, al dividir el trabajo secuencial en diferentes procesos que operan de forma concurrente sobre diferentes partes de la matriz.

Una vez presentada la aplicación con la que se realizará la evaluación, haremos un cálculo del *speedup* de la aplicación, con el objetivo de mostrar cuánto es más rápido el programa al ejecutarlo de forma paralela en n procesadores. Para la realización de la evaluación utilizaremos siempre el programa *Gradiente* con un número de iteraciones fijado a 5000.

Número de procesos	Tiempo secuencial	Tiempo paralelo	Speedup
1	1446 seg	1446 seg	**1**
2	1446 seg	769,65 seg	**1,88**
4	1446 seg	430,1 seg	**3,36**
8	1446 seg	283,29 seg	**5,1**

Tabla 32: Cálculo del Speedup de la aplicación paralela

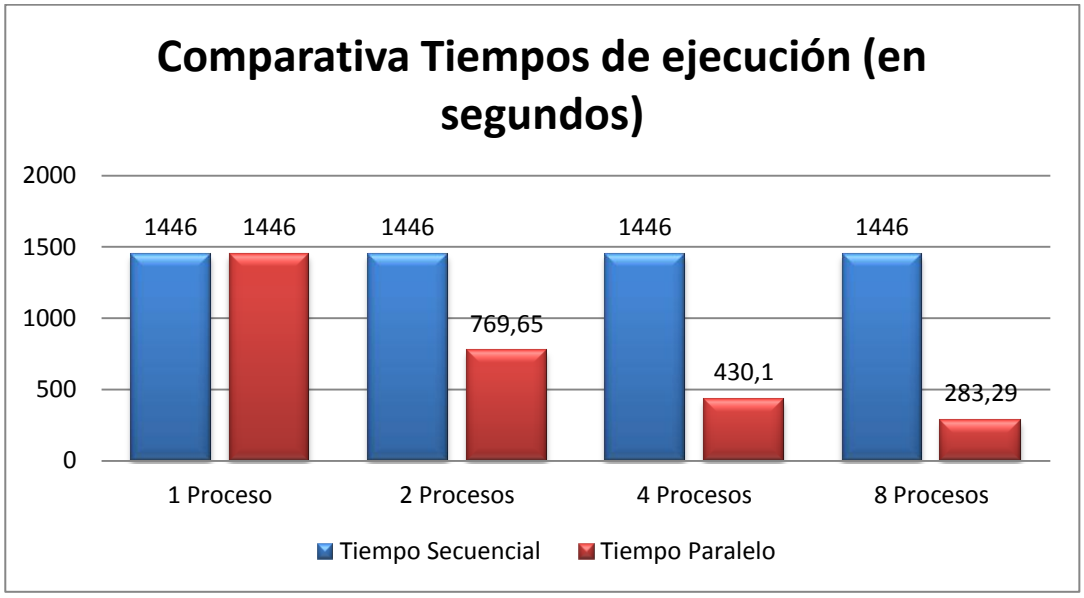


Tabla 33: Comparativa tiempos de ejecución aplicación paralela

Con los datos obtenidos en la Tabla 33: Comparativa tiempos de ejecución aplicación paralela, se demuestra que la aplicación sufre una aceleración al aumentar el número de procesos paralelos, y un aumento del *speedup* de la misma.

6.2 – Definición de Pruebas

Una vez introducida la aplicación paralela, se definen las diferentes evaluaciones que se realizarán. Una vez conocida la arquitectura del entorno de ejecución en el clúster *Tucan* de la Universidad Carlos III de Madrid, se aplican las siguientes restricciones:

- Para todas las evaluaciones la aplicación *Gradiente* se ejecutará realizando un número de 5000 iteraciones.
- El número de procesos de la aplicación Gradiente será de 8, repartidos en 2 nodos del clúster, ubicando cada proceso en uno de los 4 procesadores que posee cada nodo.

A partir de estas restricciones se definen las pruebas que se realizarán:

- **Bloque 1: Ejecución junto a Módulo de Comunicaciones de Red**
  - Comunicaciones entre un par de procesos
  - Comunicaciones entre dos pares de procesos
  - Comunicaciones entre 4 pares de procesos
- **Bloque 2: Ejecución junto a Módulo de Memoria**
  - Un proceso realizando lecturas y escrituras sobre un volumen de datos de 512 Mb
  - Dos procesos realizando lecturas y escrituras sobre un volumen de datos de 512 Mb
  - Cuatro procesos realizando lecturas y escrituras sobre un volumen de datos de 512 Mb
  - Ocho procesos realizando lecturas y escrituras sobre un volumen de datos de 512 Mb
- **Bloque 3: Ejecución junto a Módulo de CPU**
  - Varios procesos utilizando el 25% de CPU
  - Varios procesos utilizando el 50% de CPU
- **Bloque 4: Ejecución junto a Módulo de CPU en entorno alternativo**
  - Un único proceso utilizando un 20 % de CPU
  - Un único proceso utilizando un 40 % de CPU
  - Un único proceso utilizando más de un 50 % de CPU
- **Bloque 5: Ejecución junto a *Evaluación* con todos los módulos integrados**

6.3 – Análisis de las Pruebas

En este apartado se realizará un análisis de los resultados extraídos de las distintas evaluaciones.

6.3.1 – Bloque 1- Ejecución junto a módulo de Comunicaciones de Red

En todas las evaluaciones del bloque 1, los procesos realizarán comunicaciones entre sí. Estas comunicaciones podrán ser de diferentes tipos configurables por el usuario. Para la realización de las distintas evaluaciones se han definido 3 tipos de comunicaciones, con los siguientes atributos:

	Tipo 1	Tipo 2	Tipo 3
Tamaño mensaje	64 bytes	32 bytes	128 bytes
Tiempo espera	1 segundo	3 segundos	5 segundos
Proceso emisor	MIN_RANK	MAX_RANK	MIN_RANK

Tabla 34: Diferentes tipos de comunicaciones entre procesos

Comunicaciones entre un par de procesos (2 procesos)

Para esta evaluación introduciremos el Módulo de Comunicaciones en 2 procesadores, uno de cada nodo, quedando el siguiente esquema físico de los nodos:

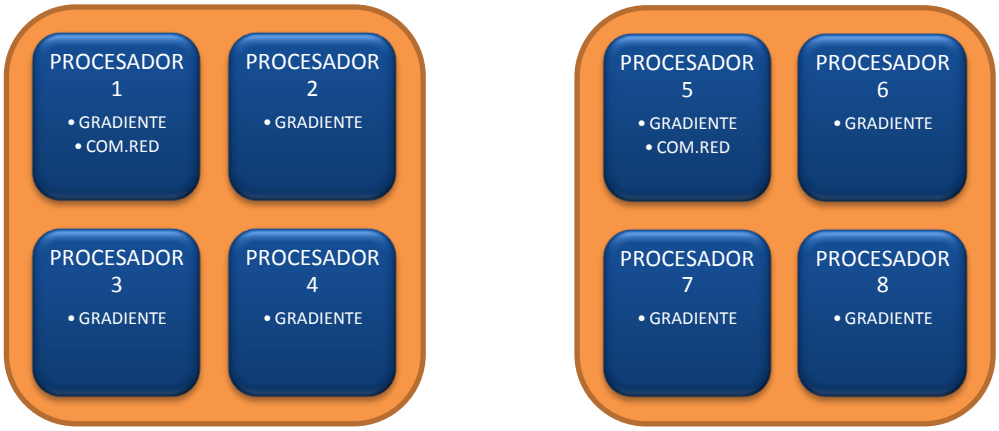


Ilustración 22: Esquema de los nodos en evaluación (1)

Según este esquema, los procesadores número 1 y 5 se comunicarán mediante llamadas punto a punto de MPI, introduciendo cargas computacionales en el sistema y en la red de comunicaciones. Ambos procesos realizarán el tipo de comunicación número 1.

Los tiempos de ejecución de los diferentes procesos son:

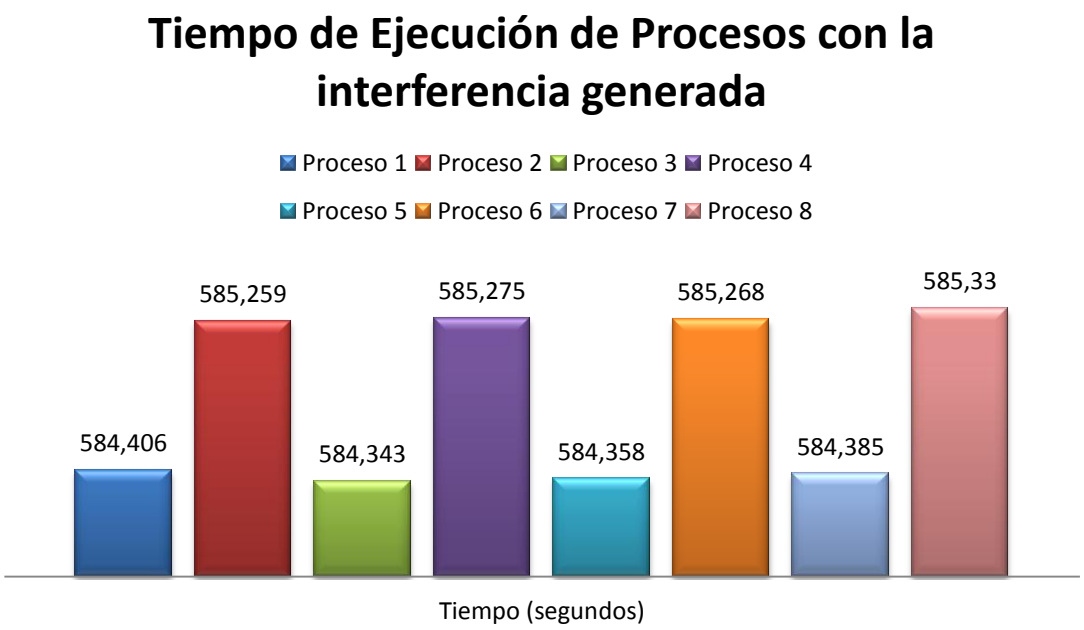


Tabla 35: Tiempo de ejecución de Gradiente con Módulo de Red (1)

En la anterior gráfica observamos los tiempos de ejecución de los diferentes procesos de la aplicación *Gradiente* en el supuesto de ejecución propuesto. El tiempo de ejecución total está definido por el proceso con mayor retardo, por lo que el tiempo total de ejecución es de **585,33** segundos.

COMUNICACIONES ENTRE DOS PARES DE PROCESOS (4 PROCESOS)

En esta evaluación se ejecuta el Módulo de Comunicaciones en 4 procesadores, dos de cada nodo, quedando el siguiente esquema físico de los nodos:

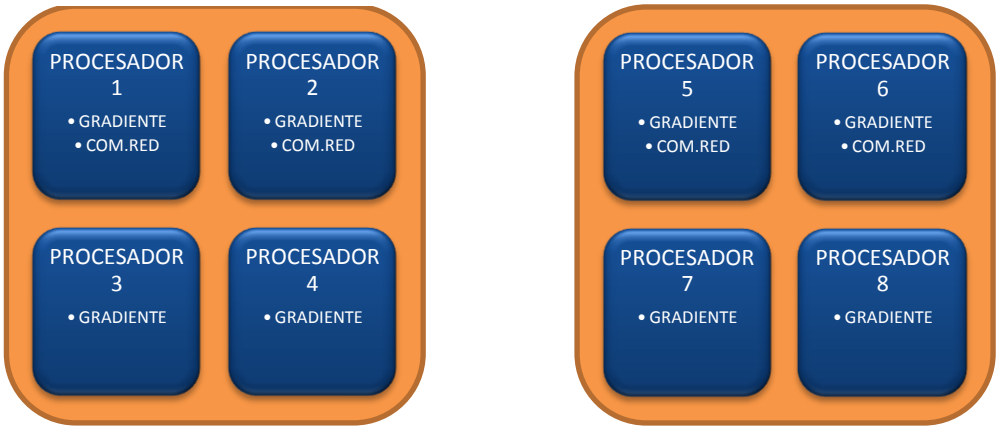


Ilustración 23: Esquema de los nodos en evaluación (2)

Según este esquema, los procesadores número 1 y 5 se comunicarán mediante llamadas punto a punto de MPI, al igual que los procesadores 2 y 6 introduciendo cargas computacionales en el sistema y en la red de comunicaciones. Los procesos 1 y 5 realizarán una comunicación de tipo 1, mientras que los procesos 2 y 6 realizarán comunicaciones de tipo 2.

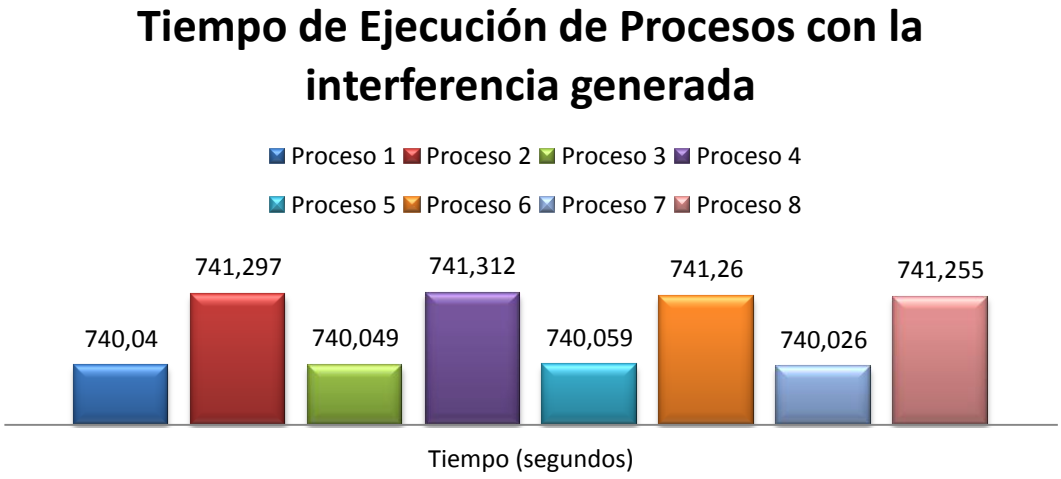


Tabla 36: Tiempo de ejecución de Gradiente con Módulo de Red (2)

Observamos los tiempos de ejecución de los diferentes procesos de la aplicación *Gradiente* para este supuesto. El tiempo máximo de ejecución de los procesos, es de **741,312** segundos, tiempo que nos dará el tiempo total de la ejecución.

COMUNICACIONES ENTRE CUATRO PARES DE PROCESOS (8 PROCESOS)

En esta evaluación introduciremos el Módulo de Comunicaciones en todos los procesadores, cuatro por nodo, quedando el siguiente esquema físico de los nodos:

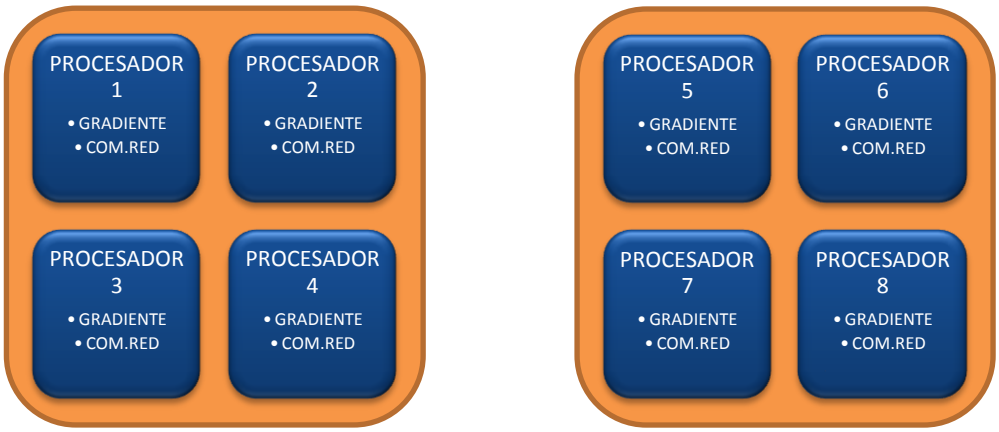


Ilustración 24: Esquema de los nodos en evaluación (3)

A partir del esquema, los pares (1,5), (2,6), (3,7) y (4,8) se comunicarán mediante llamadas punto a punto de MPI generando cargas computacionales en el sistema y en la red de comunicaciones.

Al igual que en el anterior escenario, los pares (1,5) y (2,6) realizarán los tipos de comunicaciones 1 y 2, respectivamente. Además, el par de nodos (3,7) establecerá una comunicación de tipo 3, mientras que el par (4,8) realizará, al igual que el par (1,5) una comunicación de tipo 1.

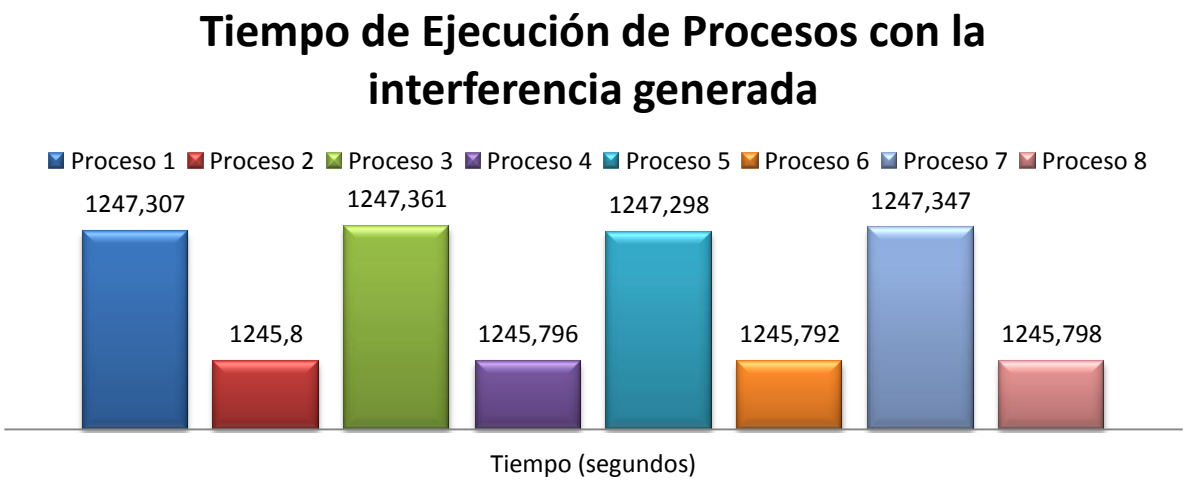


Tabla 37: Tiempo de ejecución de Gradiente con Módulo de Red (3)

Observamos los tiempos de ejecución de los diferentes procesos de la aplicación *Gradiente* para el escenario actual, obteniendo un tiempo de ejecución de **1247,361** segundos.

RESUMEN DEL BLOQUE 1:

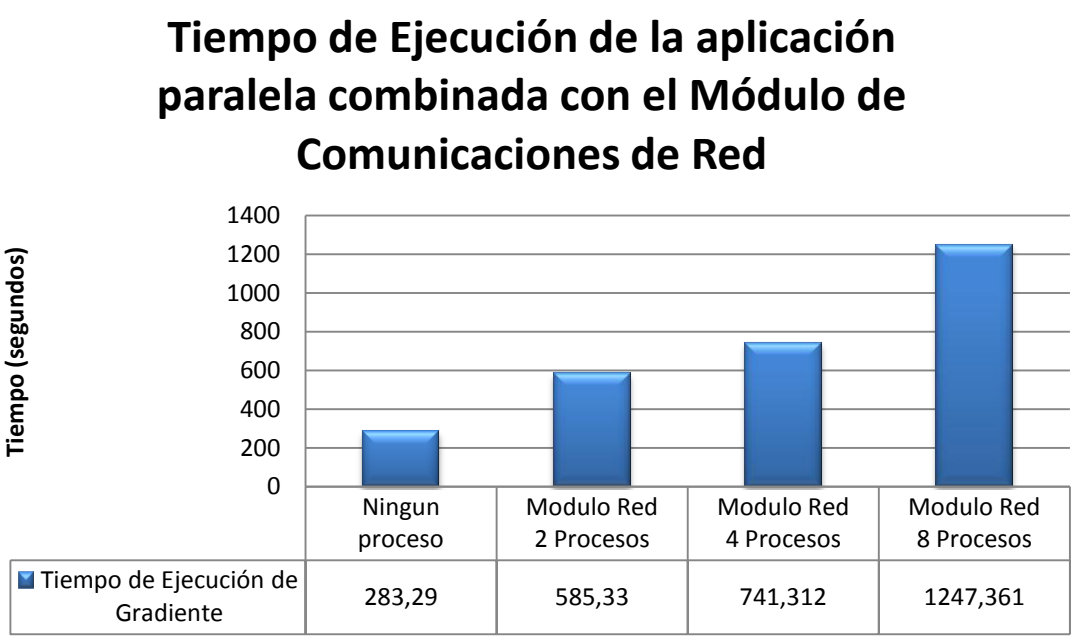


Tabla 38: Tiempo de ejecución de Gradiente con Módulo de Red (4)

Observamos en esta gráfica la comparativa de tiempos de ejecución de la aplicación. Se observa que se produce un aumento en el tiempo de ejecución al introducir cargas computacionales en el sistema a través de comunicaciones de red, realizadas mediante llamadas punto a punto del estándar MPI.

Además de producirse un aumento en el tiempo de ejecución debido a las interferencias generadas en la red de comunicaciones entre los distintos procesadores, también habrá un incremento en la utilización de la CPU para la ejecución de estos procesos. Esta utilización de tiempo de CPU de los diferentes procesos también influirá de manera importante en el retardo de la ejecución de la aplicación paralela, ya que tendrá que compartir la CPU e invertirá más tiempo en realizar todas las tareas.

6.3.2 – BLOQUE 2 – EJECUCIÓN JUNTO A MÓDULO DE MEMORIA

Para las distintas evaluaciones, el volumen de datos en el que operará cada proceso será de 512 MB, realizando operaciones de lectura y escritura. El porcentaje especificado de lecturas que se realizarán será del 30 %, mientras que el 70 % de las operaciones serán escrituras en memoria.

UN PROCESO REALIZANDO OPERACIONES DE MEMORIA SOBRE UN VOLUMEN DE DATOS

En esta evaluación introduciremos el Módulo de Memoria solamente en uno de los procesadores, quedando el siguiente esquema físico de los nodos:

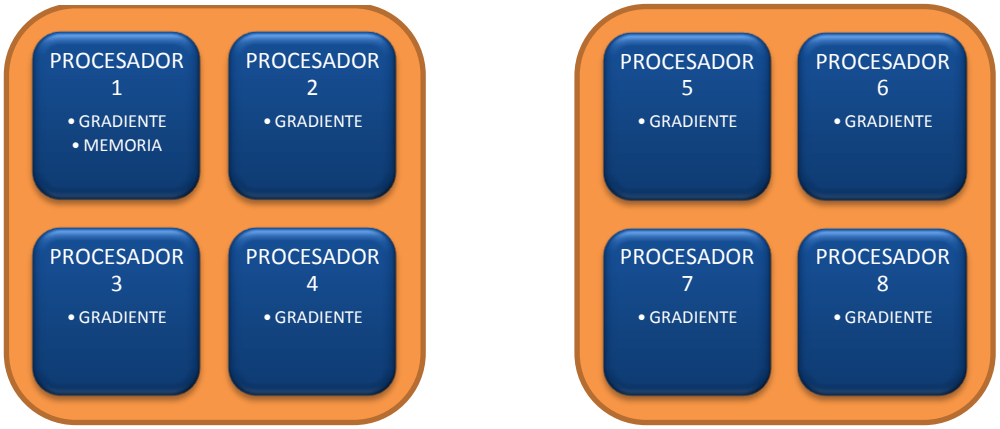


Ilustración 25: Esquema de los nodos en evaluación (4)

A partir del esquema, se observa que el proceso instalado en el procesador número 1, generará cargas computacionales en el sistema, realizando operaciones de lectura y escritura en memoria que generarán cargas entre la CPU y la memoria.

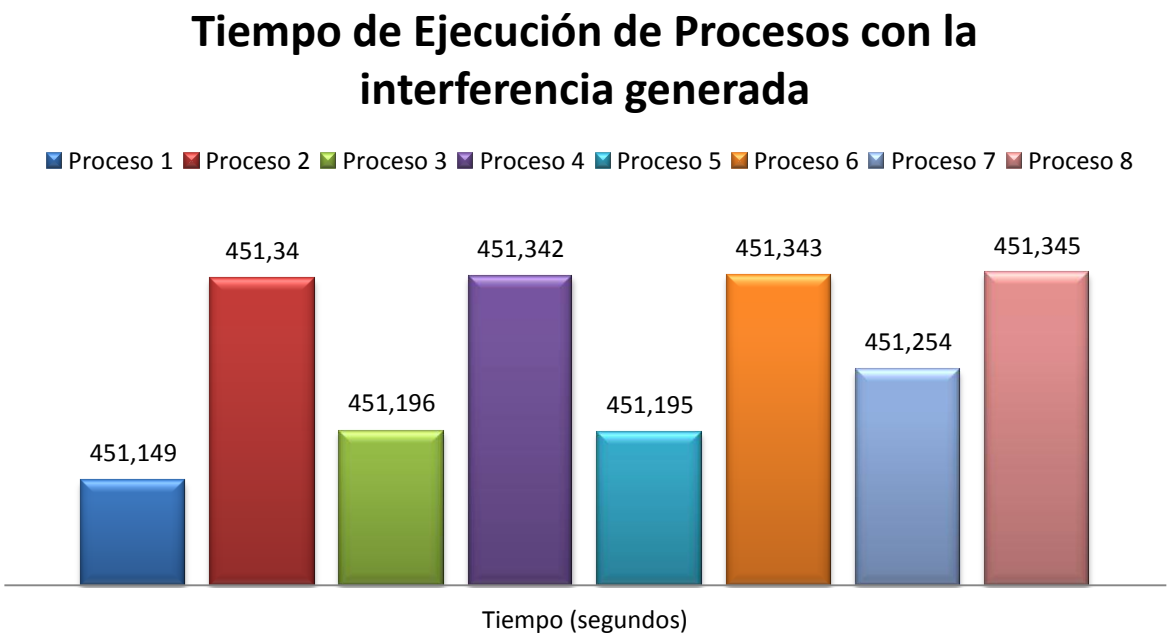


Tabla 39: Tiempo de ejecución de Gradiente con Módulo de Memoria (1)

Observamos los tiempos de ejecución de los diferentes procesos de la aplicación Gradiente en este nuevo escenario. El tiempo máximo, y por lo tanto, tiempo total de ejecución es de **451,345** segundos.

DOS PROCESOS REALIZANDO OPERACIONES DE MEMORIA SOBRE UN VOLUMEN DE DATOS

En esta evaluación introduciremos el Módulo de Memoria en dos procesadores, uno de cada nodo, quedando el siguiente esquema físico de los nodos:

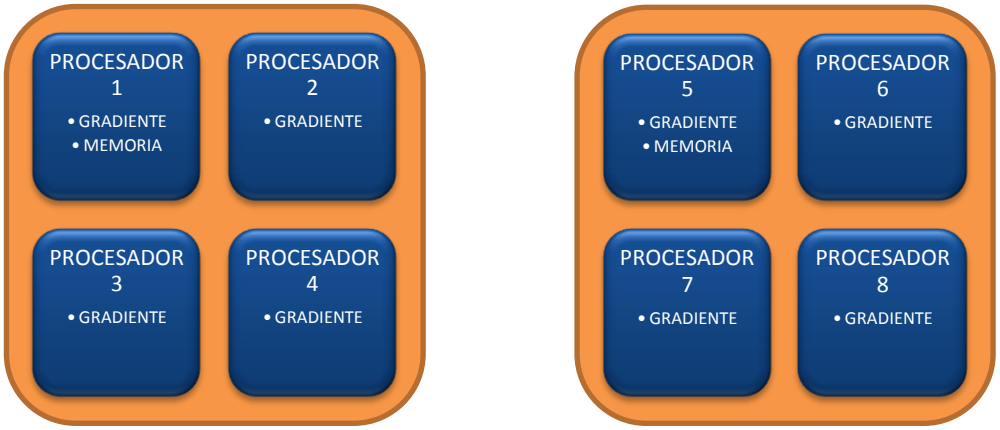


Ilustración 26: Esquema de los nodos en evaluación (5)

A partir del esquema, se observa que ambos procesos, situados en distintos nodos, generarán cargas computacionales entre la CPU y la memoria, realizando operaciones de lectura y escritura sobre el volumen de datos especificado.

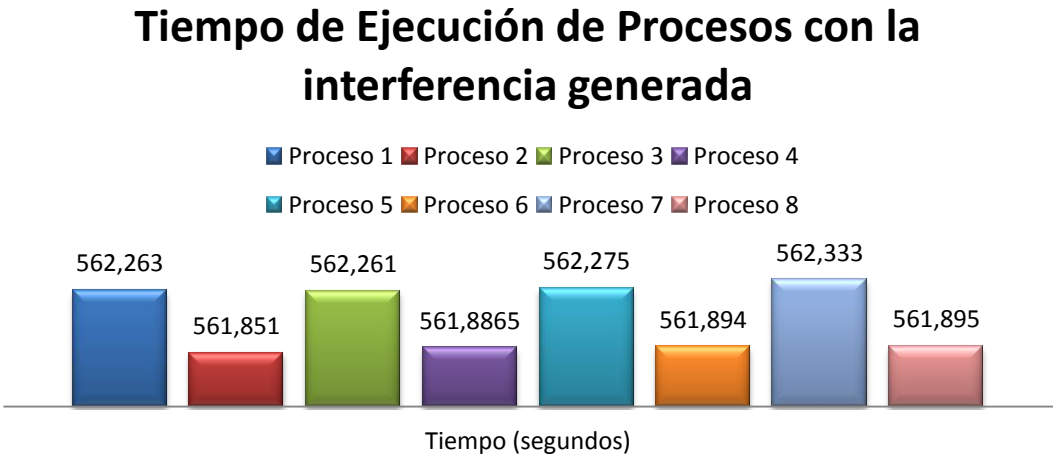


Tabla 40: Tiempo de ejecución de Gradiente con Módulo de Memoria (2)

Observamos los tiempos de ejecución de los diferentes procesos de la aplicación Gradiente, se obtiene un tiempo máximo de ejecución de **562,333** segundos.

CUATRO PROCESOS REALIZANDO OPERACIONES DE MEMORIA SOBRE UN VOLUMEN DE DATOS

En esta evaluación introduciremos el Módulo de Memoria en cuatro de los procesadores, dos por cada nodo, dando como resultado el siguiente esquema físico de los nodos:

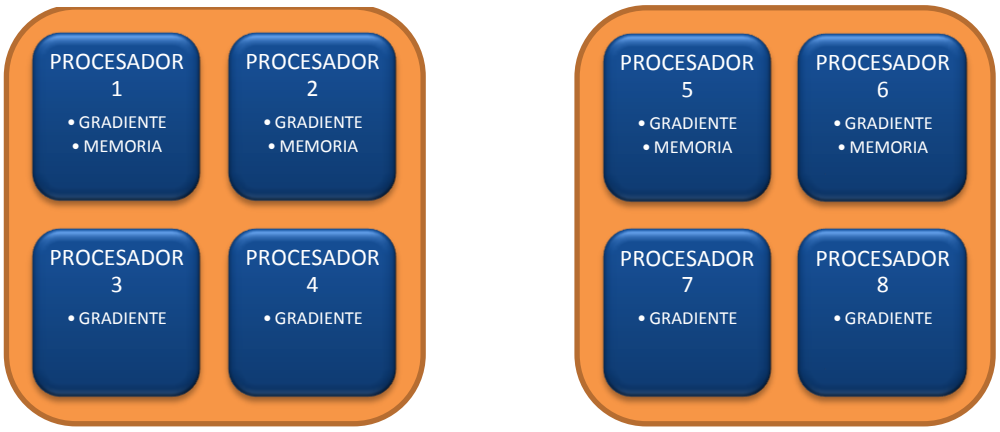


Ilustración 27: Esquema de los nodos en evaluación (6)

A partir del esquema, concluimos que los 4 procesos (1, 2, 5 y 6) generarán cargas computacionales en el sistema, realizando operaciones de lectura y escritura sobre la memoria del mismo.

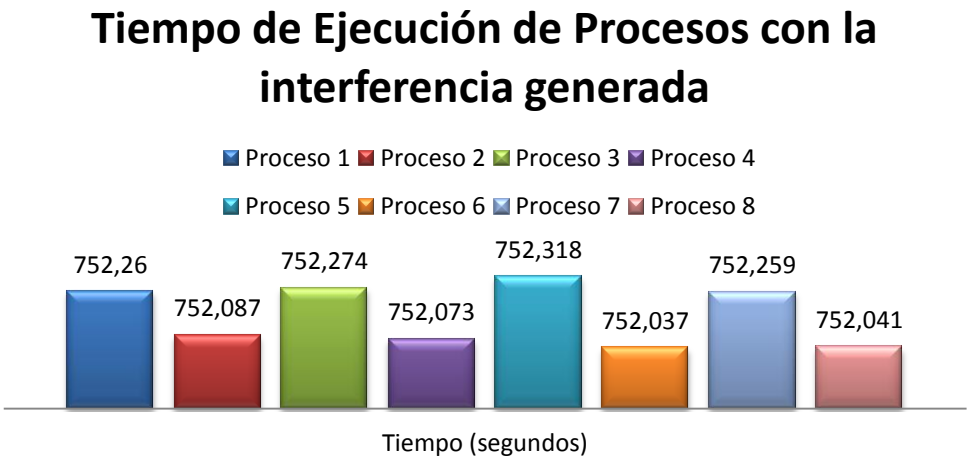


Tabla 41: Tiempo de ejecución de Gradiente con Módulo de Memoria (3)

Observamos los tiempos de ejecución de los 8 procesos de la aplicación Gradiente, el tiempo máximo de ejecución de los procesos es de **752,318** segundos, tiempo que nos indicará el tiempo de ejecución del programa.

OCHO PROCESOS REALIZANDO OPERACIONES DE MEMORIA SOBRE UN VOLUMEN DE DATOS

En esta evaluación introduciremos el Módulo de Memoria en los 8 procesadores (cuatro por nodo), obteniendo el siguiente esquema físico de los nodos:

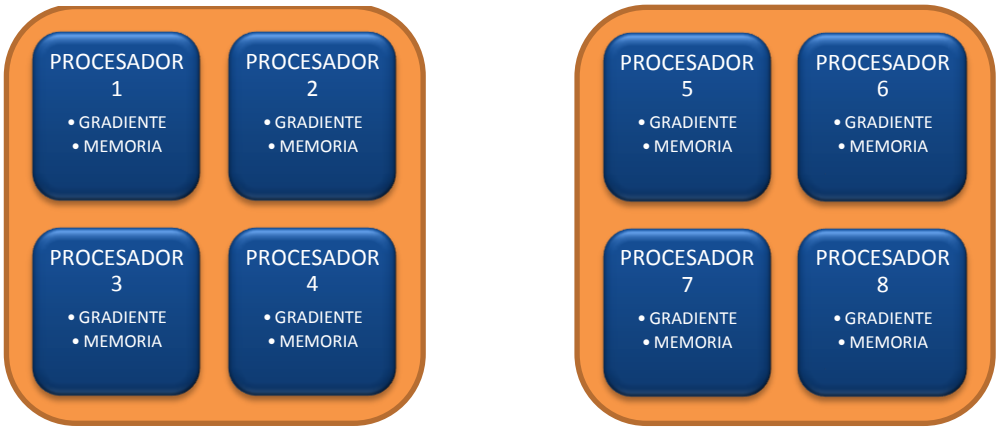


Ilustración 28: Esquema de los nodos en evaluación (7)

Con el escenario presentado en la Ilustración 30, los 8 procesos generarán cargas computacionales en el sistema, realizando operaciones, al igual que en los anteriores supuestos, de lectura y escritura sobre la memoria del mismo.

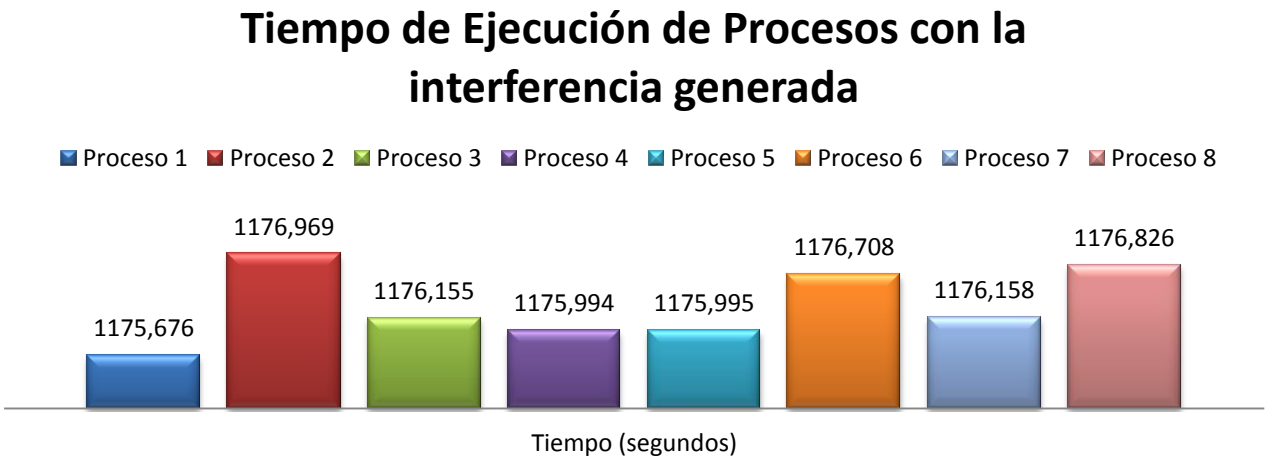


Tabla 42: Tiempo de ejecución de Gradiente con Módulo de Memoria (4)

Observando los tiempos de ejecución de los diferentes procesos de la aplicación *Gradiente*, el tiempo total de ejecución de la misma es de **1176,969** segundos.

RESUMEN DEL BLOQUE 2:

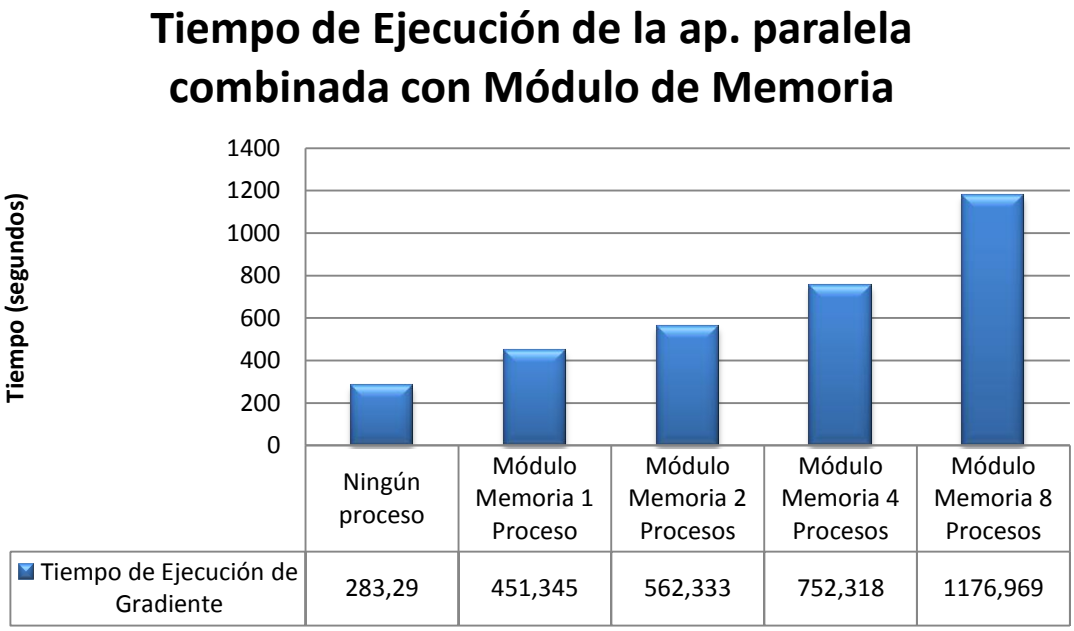


Tabla 43: Tiempo de ejecución de Gradiente con Módulo de Memoria (5)

Observamos en la comparativa de tiempos de ejecución de la aplicación. Se observa que se produce un aumento en el tiempo de ejecución al introducir otros procesos que introducen cargas computacionales en el sistema, realizando operaciones de lectura y escritura en memoria sobre un cierto volumen de datos.

Al igual que ocurre en el primer bloque de pruebas, la realización de estas operaciones, además de incrementar el tráfico entre la CPU y la Memoria, implicará un incremento del uso de cada CPU, por lo que este factor también es importante para comprender el aumento en el tiempo de ejecución de la aplicación paralela.

6.3.3 – BLOQUE 3 – EJECUCIÓN JUNTO A MÓDULO DE CPU

En este bloque se analiza el impacto que tiene la introducción de cargas computacionales en el sistema, en este caso consumiendo tiempo de CPU, de uno o varios procesos.

De igual manera que en los anteriores bloques, el escenario base será la ejecución del programa *Gradiente* en 8 procesadores, mientras que se analiza el efecto que tiene la introducción de procesos que utilizan un porcentaje específico de tiempo de uso del procesador en 1, 2, 4 y 8 procesadores.

VARIOS PROCESOS UTILIZANDO UN 25% DE CPU

Los resultados obtenidos tras la evaluación de todos los supuestos son los siguientes:

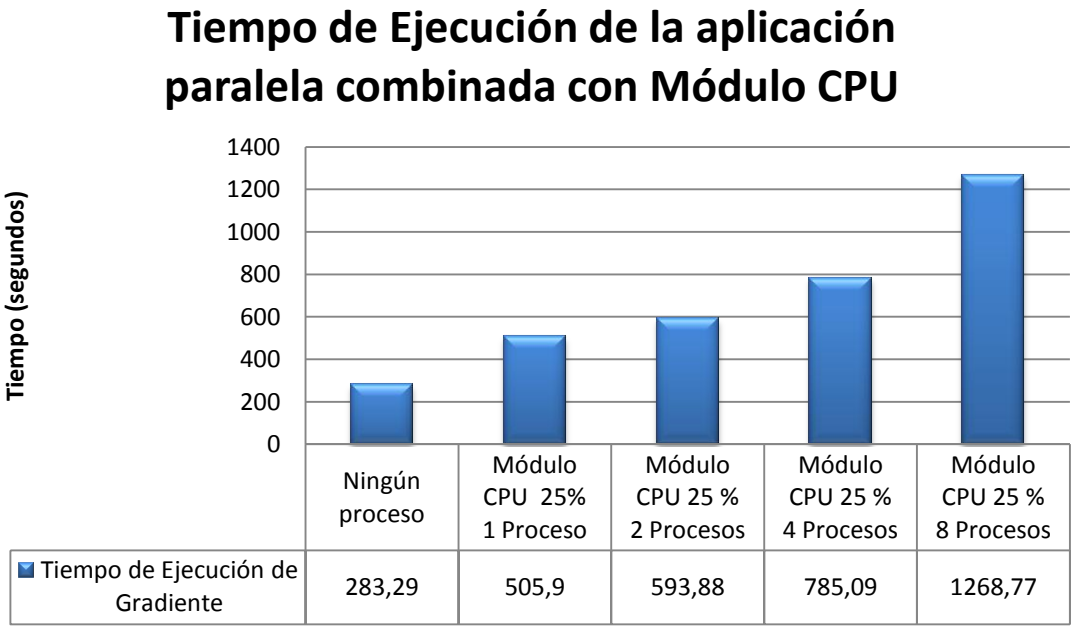


Tabla 44: Tiempo de ejecución de Gradiente con Módulo de CPU (1)

En la gráfica anterior se puede observar el aumento del tiempo de ejecución producido debido a las que se generan al aplicar cargas computacionales al procesador durante un 25 % del tiempo.

VARIOS PROCESOS UTILIZANDO UN 50% DE CPU

Los resultados obtenidos para esta evaluación son:

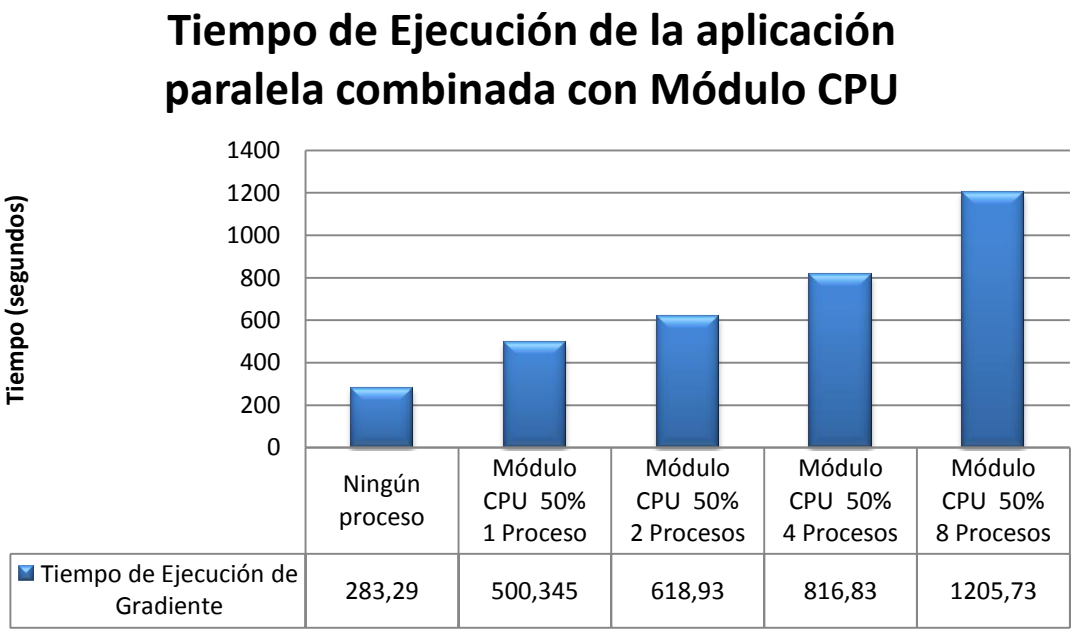


Tabla 45: Tiempo de ejecución de Gradiente con Módulo de CPU (2)

En la gráfica anterior se puede observar, de igual manera que la anterior evaluación, el aumento del tiempo de ejecución del programa *Gradiente* debido a aplicación de cargas computacionales al procesador, con un porcentaje de utilización del mismo del 50 %.

RESUMEN DEL BLOQUE 3:

Como resumen de este bloque podemos comentar que en los dos supuestos evaluados, se realiza, como se esperaba, un aumento en el tiempo de ejecución del programa *Gradiente* al aumentar el número de procesos que introducen carga computacional en el procesador.

En este bloque de pruebas obtenemos las siguientes conclusiones:

- No hay una diferencia significativa entre definir un límite de utilización del 25 % y uno del 50 %.
- No es posible la asignación de un límite de utilización de CPU superior al 50 %.

Ambas conclusiones se obtienen a partir del siguiente razonamiento:

El gestor de trabajos del clúster asigna los diferentes procesos a procesadores. El proceso Gradiente se ejecuta en los 8 procesadores, y al ejecutar el Módulo de CPU en cualquiera de ellos, se realizará una compartición de CPU. El propio gestor de trabajos presumiblemente repartirá el tiempo de CPU entre los dos procesos que convivan en el mismo procesador, y además, el reparto será equitativo, motivo por el cual no es posible alcanzar un límite de utilización de CPU superior al 50 %.

Tras consultar a los administradores del sistema y documentación en la red no ha sido posible clarificar este punto de forma completa y precisa.

Debido a esto, las pruebas se han realizado con límites del 25 % y el 50 %. Se realizará un bloque extra de pruebas con el objetivo de complementar este análisis.

6.3.4 – Bloque 4 – Ejecución junto a módulo de CPU en entorno alternativo

En este nuevo bloque de pruebas, y con el fin de ampliar el estudio del impacto de consumo de la CPU en el rendimiento del Gradiente se ha realizado una nueva evaluación, sobre una diferente plataforma. La plataforma en la que se ha realizado la evaluación es la especificada en el apartado 5.1.1 del documento, en uno de los equipos de desarrollo utilizados para la realización de las pruebas unitarias.

El objetivo de este bloque es el mismo que el del bloque anterior, pero en este caso solamente se realizará la evaluación sobre un único procesador. Por lo tanto, el escenario es el siguiente:

- Un único procesador
- Ejecución del Gradiente con un número de 1000 iteraciones.
- Los programas Gradiente y Generador_Cargas compartirán CPU y se analizará la compartición de recursos realizada, así como el posible aumento del tiempo total de la rutina de ejecución del programa Gradiente en caso de que la ejecución sea la deseada.

Un único proceso utilizando un 20% de CPU

Para realizar esta evaluación se lanza, en el mismo procesador que el programa Gradiente, el módulo de CPU, con un límite de utilización establecido en el 20 %. A continuación se adjunta una ilustración en la que se aprecia el porcentaje de uso de ambas aplicaciones:

```
top - 20:42:13 up 45 min,  4 users,  load average: 2.38, 1.76, 1.19
Tasks: 154 total,   4 running, 150 sleeping,   0 stopped,   0 zombie
Cpu(s): 82.5%us, 17.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1025156k total,   925140k used,   100016k free,     7172k buffers
Swap:  1046524k total,    14936k used,  1031588k free,   400348k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
18144 juanfran  20   0  233m 226m 1292 R 45.4 22.6   1:48.07 gradient
20390 juanfran  20   0  172m  576  412 S 19.2  0.1   0:52.82 cpu
```

Ilustración 29: Captura de pantalla de comando top (1)

En la ilustración se puede observar que el porcentaje de utilización de la CPU del Gradiente no supera el 50 %, de la misma manera que ocurría en la evaluación del bloque 3. El porcentaje alcanzado por el módulo de CPU se comprueba que es correcto.

UN ÚNICO PROCESO UTILIZANDO UN 40% DE CPU

Para realizar esta evaluación se lanza, en el mismo procesador que el programa Gradiente, el módulo de CPU, con un límite de utilización establecido, en este caso, en el 40 %. A continuación se adjunta una ilustración en la que se aprecia el porcentaje de utilización de ambas aplicaciones:

```
top - 21:05:14 up  1:08,  4 users,  load average: 1.59, 0.62, 0.68
Tasks: 151 total,   3 running, 148 sleeping,   0 stopped,   0 zombie
Cpu(s): 92.1%us,  7.9%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1025156k total,   918264k used,   106892k free,     8544k buffers
Swap:  1046524k total,    16312k used,  1030212k free,   388584k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 8155 juanfran  20   0  233m 226m 1292 R 48.0 22.6   0:36.58 gradient
 8003 juanfran  20   0 20676  500  412 S 37.7  0.0   0:32.33 cpu
```

Ilustración 30: Captura de pantalla del comando top (2)

En este caso, se aprecia el mismo resultado que en la evaluación anterior, cambiando únicamente el porcentaje de uso del módulo de CPU, que se acerca al 40 % establecido. La utilización del programa Gradiente continúa sin superar el 50 % de la utilización de la CPU.

UN ÚNICO PROCESO UTILIZANDO UN PORCENTAJE DE UTILIZACIÓN DE LA CPU SUPERIOR AL 50%

Para realizar esta evaluación se lanza, en el mismo procesador que el programa Gradiente, el módulo de CPU, con un límite de utilización superior al 50 %. Para la realización de esta evaluación se establecerá en el 60 %. A continuación se adjunta una ilustración en la que se aprecia el porcentaje de utilización de ambas aplicaciones:

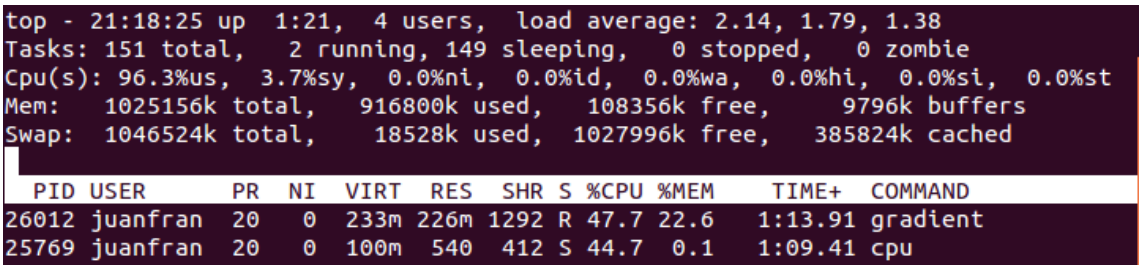


Ilustración 31: Captura de pantalla del comando top (3)

Tal y como ya ocurrió en las evaluaciones del bloque 3, se observa que por motivos que no han podido ser detectados, no es posible alcanzar un límite superior al 50 %.

RESUMEN DEL BLOQUE 4:

En los dos supuestos evaluados, se da, como se esperaba, un aumento en el tiempo de ejecución del programa *Gradiente* al introducir cargas computacionales en el procesador mediante el módulo de CPU. Los tiempos obtenidos en la ejecución del programa Gradiente son los siguientes:

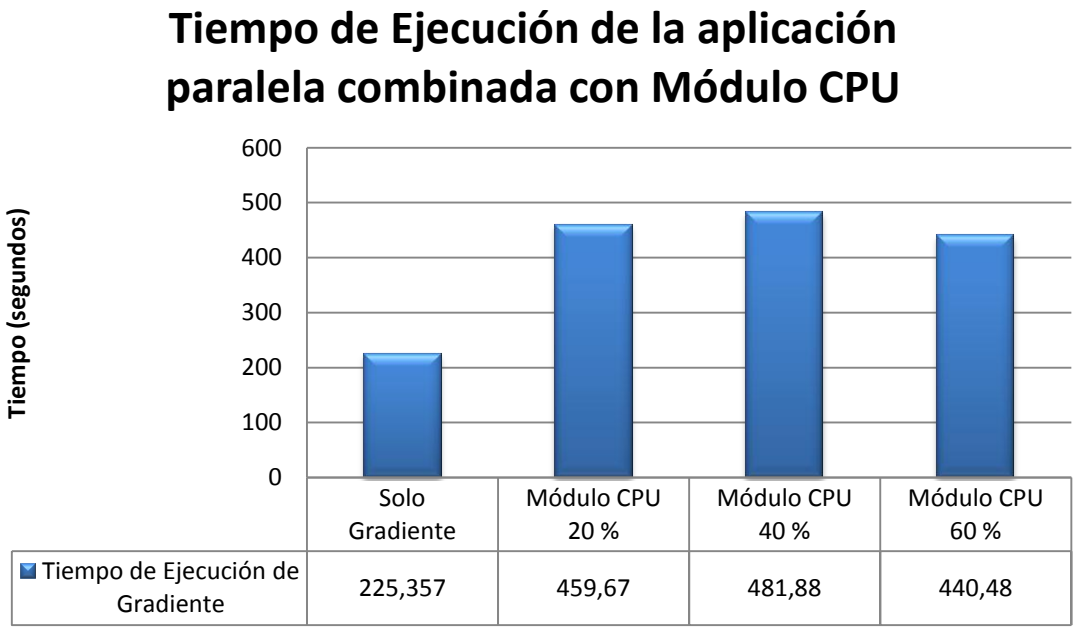


Tabla 46: Tiempo de ejecución de Gradiente con Módulo de CPU (3)

Observando los tiempos de ejecución del programa Gradiente, y teniendo en cuenta las observaciones realizadas sobre su ejecución, es normal que en todas las evaluaciones conjuntas con el módulo de CPU se obtenga un tiempo de ejecución similar, ya que el proceso *Gradiente* estará utilizando, como máximo, un 50 % del procesador.

6.3.5 – Bloque 5 – Ejecución junto a Evaluación con todos los Módulos integrados

Después de analizar el impacto que realiza en la ejecución del programa *Gradiente* cada uno de los módulos desarrollados en el programa *Evaluación,* se hace un estudio sobre cómo afecta la ejecución del programa completo al tiempo de ejecución del programa *Gradiente*.

Como ya se hiciera en los anteriores bloques, el escenario base será la ejecución del programa *Gradiente* en 8 procesadores, mientras que se analiza el efecto que tiene la introducción del programa desarrollado en este proyecto en 2, 4 y 8 procesadores.

La configuración que se aplicará será la utilizada en los anteriores bloques de pruebas específicas, es decir, el módulo de memoria operará sobre un volumen de datos de 512 Mb y las comunicaciones entre procesos serán las mismas que fueron definidas en el bloque 1, dependiendo del número de procesos que intervengan. Como configuración del módulo de CPU, se aplica un porcentaje límite de utilización de la CPU del 20 %.

Debido al problema comentado en el Resumen de los bloques 3 y 4, en este caso se aplica un porcentaje límite del 20 %. Recordando el escenario del Bloque 3, el módulo de CPU se ejecuta en el mismo procesador que el proceso *Gradiente,* realizando una compartición de CPU equitativa entre los dos procesos. En este caso, el escenario es diferente: El programa Evaluación se compone de tres hilos de ejecución o threads, por lo que el número de tareas que tendrán que ejecutarse en cada procesador será de cuatro. Esto explica, que para este bloque, el porcentaje límite de utilización de la CPU que pueda alcanzar la aplicación sea del 25 %.

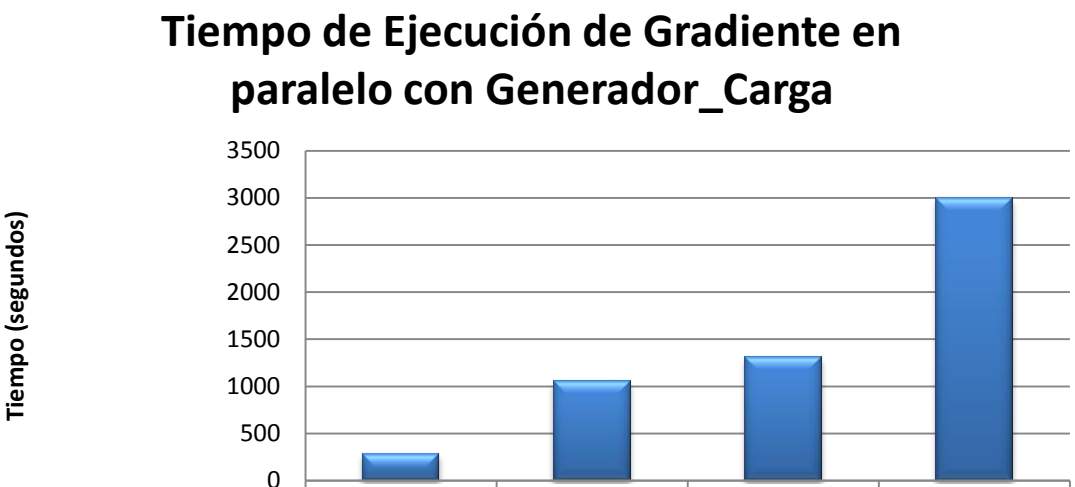


Tabla 47: Tiempo de ejecución de Gradiente junto a Generador_Carga

Analizando los resultados obtenidos, observamos que el tiempo de ejecución aumenta de una mayor forma que en las pruebas individuales de los módulos.

Este comportamiento es debido al fenómeno explicado anteriormente. Al igual que el thread que ejecuta el módulo de CPU solamente podrá utilizar un máximo de tiempo de CPU del 25 %, las demás tareas se encuentran en idéntica situación. Debido a esto, se obtiene un aumento del tiempo de ejecución de la aplicación *Gradiente* tan grande, debido a que los procesos que comparten procesador con la aplicación desarrollada en el proyecto solamente se ejecutan el 25 % del tiempo, trasladando el retardo en la ejecución al resto de procesos.

7 – PRESUPUESTO

En el presente apartado se realizará un presupuesto detallado del proyecto, dividiendo el análisis en varios subgrupos, que serán conjuntados para indicar el coste final.

El proyecto ha sido desarrollado íntegramente por el alumno que realiza el Trabajo de Fin de Grado, y por lo tanto, ha realizado todas las fases del desarrollo del mismo.

Como ya se indicó en el apartado 4.2 – Planificación temporal, la fecha de inicio del proyecto se fija el **1 de Febrero de 2012**, mientras que la fecha de fin se ha estimado para el **5 de Septiembre de 2012**.

El tiempo estimado de trabajo para la realización del proyecto es de **300 horas**, pudiendo ampliarse las horas de trabajo en caso de que fuese necesario. Esta decisión fue tomada junto al tutor del proyecto en una de las reuniones de seguimiento del proyecto. Las horas que se superen a partir de las máximas estimadas tendrán el mismo coste.

7.1 – COSTES DE PERSONAL

A continuación podemos apreciar una tabla en la que se especificará el coste total de cada fase de desarrollo del proyecto, indicando el número de horas dedicadas a cada una, y el coste por cada hora de trabajo.

Fase desarrollo	Coste/Hora	Horas Totales	Coste Total
Análisis	25.00 €	35	875 €
Diseño	25.00 €	45	1.125 €
Implementación	25.00 €	150	3.750 €
Pruebas	25.00 €	80	2.000 €
Documentación	25.00 €	60	1.500 €
TOTAL		**370**	**9.250 €**

Tabla 48: Costes de Personal

7.2 – Costes de Hardware y Software

En la siguiente tabla mostraremos el coste del equipo de desarrollo, el software utilizado, y también el coste de los equipos de pruebas utilizados. Para los equipos suponemos un periodo de amortización de 3 años, ya que la tecnología avanza muy rápidamente y el equipo puede quedar obsoleto en dicho periodo de tiempo.

En cuanto al software, se establece un periodo de amortización de 5 años, pero en nuestro caso no aplica, ya que el único software no gratuito que se ha utilizado va incluido en el precio de la compra del equipo de desarrollo.

Objeto	Unidades	Coste	Subtotal
Equipo desarrollo	1	900 €	175,00 €
Equipos pruebas	2	2.500 €	487,00 €
Windows 7 Professional	1	Incluido en coste equipo de desarrollo	0,00 €
Ubuntu	1	Gratuito	0,00 €
Microsoft Office 2007	1	Incluido en coste equipo de desarrollo	0,00 €
Otros			0,00 €
TOTAL			**662 €**

Tabla 49: Costes de Hardware y Software

7.3 – Costes Indirectos

Como costes indirectos introduciremos los gastos en la conexión a Internet, gastos de luz y tinta y papel para la impresión de la documentación.

Elemento	Unidades	Coste unidad	Total
Conexión a Internet	7 Meses	30,00 €	210,00 €
Luz	7 Meses	45,00 €	315,00 €
Papel	1	15,00 €	15,00 €
Tinta impresión	1	10,00 €	10,00 €
TOTAL			**550 €**

Tabla 50: Costes indirectos

7.4 – Coste Total

Después de sumar todos los costes anteriores (de personal, de hw/sw e indirectos), aplicaremos un porcentaje de riesgo del 5% para posibles complicaciones en el desarrollo del proyecto y un 18% de beneficio para el desarrollador del mismo.

Concepto	Coste
Costes de Personal	9.250 €
Costes de Hardware y Software	662 €
Costes Indirectos	550 €
SUBTOTAL 1	**10.462 €**
Riesgo (5%)	523 €
Beneficio (18%)	1.884 €
SUBTOTAL 2	**12.869 €**
IVA (21%)	2.702 €
TOTAL	**15.571 €**

Tabla 51: Coste Total

El coste total presupuestado para la realización del proyecto es de:

15.571 €

8 – CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se realizará un breve resumen del trabajo realizado durante el desarrollo del proyecto, y se tratará de dar respuesta o analizar la situación final de aquellos objetivos marcados al principio del presente documento. Además, se comentarán las conclusiones principales obtenidas por el alumno tras la realización de este proyecto, así como los principales problemas encontrados.

Finalmente, se expondrá un listado de posibles trabajos futuros que pueden ser interesantes para complementar o completar al proyecto.

Cuando se inició el proyecto, se enumeraron una serie de objetivos que se pretendían cumplir al finalizar el desarrollo del mismo. El primero de estos objetivos era la familiarización con la interfaz de paso de mensajes MPI. Este primer objetivo se ha cumplido con creces, ya que se ha desarrollado una aplicación MPI capaz de generar una serie de cargas computacionales, de forma controlada, y capaces de variar el tráfico tanto de red como de memoria, y la carga de la CPU de los diferentes nodos.

Para la evaluación de los resultados finales, se han realizado diferentes pruebas. En primer lugar, se ha probado la aplicación desarrollada, de forma aislada, con el objetivo de comprobar su correcto funcionamiento. Una vez confirmado éste, se ha realizado una evaluación en un entorno real (un clúster), analizando el impacto que tiene sobre la plataforma la introducción de diferentes cargas de cómputo que puedan generar interferencias en la ejecución de otras aplicaciones en dichos nodos.

Echando la vista atrás, se puede comprobar que todos aquellos objetivos marcados al inicio del desarrollo del proyecto han sido superados con éxito.

Además, los resultados finales son bastante buenos, y permiten al grupo de investigación abrir nuevas vías de estudio para, por ejemplo, poder desarrollar técnicas que mejoren la eficiencia de aplicaciones que deban adaptarse a condiciones cambiantes en el entorno de ejecución de la misma.

Uno de los principales problemas o inconvenientes que se han tenido para la realización del proyecto ha sido la interpretación de los resultados, debido a la cantidad de factores que se deben considerar en la misma. El principal problema se ha tenido a la hora de intentar entender la forma en la que el planificador de tareas del

sistema operativo realiza sus funciones, y la forma en la que reparte los recursos a los diferentes procesos.

Para el alumno, la realización de este proyecto ha sido muy gratificante ya que ha supuesto una forma de aplicar muchos de los conocimientos teóricos adquiridos durante la realización de la carrera, sobre todo en aspectos de planificación.

Por último, se enumerarán una serie de propuestas como trabajos futuros que completen o complementen a este proyecto:

- Mejora en la gestión de parámetros de la aplicación. Se propone la creación de un método de introducción de parámetros, que no implique cambios en el código fuente de la aplicación.
- Mayor automatización en la ejecución de la aplicación. Se sugiere la creación de un sistema de control de la aplicación, desde el que se puedan realizar operaciones de forma más sencilla e intuitiva.
- Investigar interacción entre el la aplicación desarrollada y el sistema operativo, concretamente en como el planificador de tareas puede afectar en la ejecución del módulo de consumo de CPU.
- Extender el estudio a otras aplicaciones y plataformas. Un ejemplo de esto sería el de realizar el estudio sobre una plataforma con una red de comunicaciones muy lenta, en la que se apreciaría un aumento en el tiempo de ejecución de las aplicaciones que hagan uso de esta característica.

9 – REFERENCIAS

[1] Web de Programación Distribuida y Paralela [En línea]

[2] Ficheros C/C++ - Archivos de acceso aleatorio [En línea]

[3] How to get %CPU usage of a process [En línea]

[4] Comandos útiles de consola | Ubuntu-es [En línea]

[5] how to get a thread's ID in c/c++ gnu, gettid() is not implemented - Ubuntu Forums [En línea]

[6] Funcion system con argumentos - psicofxp.com [En línea]

[7] nanosleep is better than sleep and usleep | c/c++ programming by examples [En línea]

[8] Cloud computing, tipos de nubes | [En línea]

[9] MPICH2 : High-performance and Widely Portable MPI [En Línea]

[10]ARCOS [En línea]

[11] Wikipedia, la enciclopedia libre [En línea]

[12] Introducción al multithreading con Pthreads [Universidad de Buenos Aires,2012]

ANEXO 1 – MANUAL DE USUARIO

Para lanzar la aplicación es necesario seguir una serie de pasos:

- Extracción de los archivos:

```
juanfran@ubuntu:~/Desktop/prueba$ ls -lart
total 12
drwxr-xr-x 3 juanfran juanfran 4096 Aug  6 17:55 ..
-rw-rw-r-- 1 juanfran juanfran 3843 Aug  6 17:58 TFG-Evaluacion.tar.gz
drwxrwxr-x 2 juanfran juanfran 4096 Aug  6 17:58 .
juanfran@ubuntu:~/Desktop/prueba$ tar -xzvf TFG-Evaluacion.tar.gz
comunicaciones.c
cpu.c
evaluacion.c
matriz.txt
memoria.c
juanfran@ubuntu:~/Desktop/prueba$ ls -lart
total 44
-rw-rw-r-- 1 juanfran juanfran   30 Feb 23 20:37 matriz.txt
-rw-rw-r-- 1 juanfran juanfran 6254 May 21 17:38 evaluacion.c
-rwxrw-r-- 1 juanfran juanfran 2747 Jul 16 20:16 cpu.c
-rw-rw-r-- 1 juanfran juanfran 6621 Jul 18 18:00 memoria.c
-rwxrw-r-- 1 juanfran juanfran 6655 Jul 25 18:33 comunicaciones.c
drwxr-xr-x 3 juanfran juanfran 4096 Aug  6 17:55 ..
-rw-rw-r-- 1 juanfran juanfran 3843 Aug  6 17:58 TFG-Evaluacion.tar.gz
drwxrwxr-x 2 juanfran juanfran 4096 Aug  6 17:58 .
juanfran@ubuntu:~/Desktop/prueba$
```

Ilustración 32: Extracción de los archivos

El archivo .tar en el que está ubicada la aplicación contiene los siguientes archivos:

- **matriz.txt**: Fichero de texto en el que encontraremos la matriz de comunicaciones. Habrá que editar este fichero para introducir modificaciones, y dependiendo del número de procesos que se quieran lanzar. Necesario para lanzar tanto el archivo *evaluacion.c* como el *comunicaciones.c*
- **evaluación.c**: Fichero en el que se incluye la aplicación completa, con los tres módulos implementados en el mismo fuente.
- **cpu.c**: Fichero en el que se encuentra el módulo de CPU.
- **memoria.c**: Fichero con el código del módulo de Memoria.
- **comunicaciones.c**: Fichero de código con el contenido del módulo de comunicaciones de red.

Se incluyen programas únicos para los diferentes módulos, para permitir al usuario la realización de una evaluación tanto conjunta, como separada de los diferentes módulos del proyecto.

- Compilación:

```
juanfran@ubuntu:~/Desktop/prueba$ mpicc.mpich2 -o evaluacion evaluacion.c
juanfran@ubuntu:~/Desktop/prueba$
juanfran@ubuntu:~/Desktop/prueba$ mpicc.mpich2 -o cpu cpu.c
juanfran@ubuntu:~/Desktop/prueba$
juanfran@ubuntu:~/Desktop/prueba$ mpicc.mpich2 -o comunicaciones comunicaciones.
c
juanfran@ubuntu:~/Desktop/prueba$
juanfran@ubuntu:~/Desktop/prueba$ mpicc.mpich2 -o memoria memoria.c
juanfran@ubuntu:~/Desktop/prueba$
juanfran@ubuntu:~/Desktop/prueba$ ls -lart
total 100
-rw-rw-r-- 1 juanfran juanfran    30 Feb 23 20:37 matriz.txt
-rw-rw-r-- 1 juanfran juanfran  6254 May 21 17:38 evaluacion.c
-rwxrw-r-- 1 juanfran juanfran  2747 Jul 16 20:16 cpu.c
-rw-rw-r-- 1 juanfran juanfran  6621 Jul 18 18:00 memoria.c
-rwxrw-r-- 1 juanfran juanfran  6655 Jul 25 18:33 comunicaciones.c
drwxr-xr-x 3 juanfran juanfran  4096 Aug  6 17:55 ..
-rw-rw-r-- 1 juanfran juanfran  3843 Aug  6 17:58 TFG-Evaluacion.tar.gz
-rwxrwxr-x 1 juanfran juanfran 12362 Aug  6 18:11 evaluacion
-rwxrwxr-x 1 juanfran juanfran  7774 Aug  6 18:11 cpu
-rwxrwxr-x 1 juanfran juanfran 12374 Aug  6 18:11 comunicaciones
drwxrwxr-x 2 juanfran juanfran  4096 Aug  6 18:12 .
-rwxrwxr-x 1 juanfran juanfran 12359 Aug  6 18:12 memoria
juanfran@ubuntu:~/Desktop/prueba$
```

Ilustración 33: Compilación de los archivos

En la ilustración anterior podemos observar el comando de compilación necesario para la compilación de los fuentes (ficheros .c) y la generación de los ejecutables. El comando para compilar es el siguiente:

mpicc.mpich2 –o ejecutable archivo_fuente.c

Una vez generados los ejecutables podremos lanzar cualquiera de las cuatro diferentes aplicaciones utilizando el comando:

mpiexec.mpich2 –np X ./ejecutable

*El valor de X corresponde con el número de procesos que se lanzarán.

ANEXO 2 - EJEMPLOS DE EJECUCIÓN DE LA APLICACIÓN

PRUEBA DEL MÓDULO DE COMUNICACIONES

Parámetros de entrada:

- Número de procesos: 5
- Matriz de comunicaciones (por filas): [0 1 0 0 3] , [1 0 0 0 3] , [0 0 0 2 0] , [0 0 2 0 0] , [3 3 0 0 0] ;

*En el apartado 3.1.1 es posible encontrar un ejemplo de matriz de comunicaciones, así como una breve descripción de sus atributos.

Resultados:

```
juanfran@ubuntu:~/Desktop/prueba$ mpiexec.mpich2 -np 5 ./comunicaciones
Soy el thread COMUNICACIONES y el PID del proceso es: 4788
Soy el thread COMUNICACIONES y el PID del proceso es: 4785
Soy el thread COMUNICACIONES y el PID del proceso es: 4789
Soy el thread COMUNICACIONES y el PID del proceso es: 4787
Soy el thread COMUNICACIONES y el PID del proceso es: 4786
0 envia a 1 de tipo 1 y tarda 0.014809 s
1 recibe de 0 de tipo 1 y tarda 0.015408 s

3 envia a 2 de tipo 2 y tarda 0.000017 s
2 recibe de 3 de tipo 2 y tarda 0.015437 s

0 envia a 4 de tipo 3 y tarda 0.015569 s
4 recibe de 0 de tipo 3 y tarda 0.020014 s
1 envia a 4 de tipo 3 y tarda 0.020186 s
4 recibe de 1 de tipo 3 y tarda 0.035244 s
```

Ilustración 34: Resultados del módulo de Comunicaciones de Red

Tal y como se indica en la matriz de comunicaciones, se realizan las siguientes comunicaciones entre procesos:

- Procesos 0 y 1 → Tipo comunicación 1
- Procesos 2 y 3 → Tipo comunicación 2
- Procesos 0 y 4 → Tipo comunicación 3
- Procesos 1 y 4 → Tipo comunicación 3

PRUEBA DEL MÓDULO DE CPU

Parámetros de entrada:

- Límite: 60 %
- Numero de procesos: 1

Resultados:

```
Myrank es 0 y mi Id de thread es 3634419456
Soy el thread CPU y el PID del proceso es: 7111
10972
El porcentaje de uso de la CPU del thread 10972 es 60.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 66.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 71.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 77.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 82.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 88.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 93.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 99.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 63.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 66.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 69.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 71.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 74.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 77.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 80.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 82.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 85.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 88.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 91.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 93.5 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 64.3 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 66.3 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 68.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 70.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 71.6 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 73.6 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 75.3 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 77.3 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 79.0 , t=16610
El porcentaje de uso de la CPU del thread 10972 es 81.0 , t=16610
                Porcentaje medio: 76.8
```

```
Myrank es 0 y mi Id de thread es 3623929600
Soy el thread CPU y el PID del proceso es: 7111
11237
El porcentaje de uso de la CPU del thread 11237 es 9.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 14.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 19.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 24.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 29.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 34.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 39.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 44.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 49.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 54.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 59.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 64.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 69.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 37.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 39.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 42.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 44.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 47.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 49.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 52.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 54.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 57.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 59.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 62.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 64.5 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 44.6 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 46.3 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 48.0 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 49.6 , t=18271
El porcentaje de uso de la CPU del thread 11237 es 51.3 , t=18271
                Porcentaje medio: 45.2
Myrank es 0 y mi Id de thread es 3613439744
Soy el thread CPU y el PID del proceso es: 7111
11424
El porcentaje de uso de la CPU del thread 11424 es 44.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 49.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 55.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 60.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 66.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 71.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 77.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 82.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 88.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 94.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 99.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 52.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 55.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 58.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 60.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 63.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 66.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 69.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 72.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 74.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 77.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 80.5 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 83.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 57.3 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 59.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 61.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 63.0 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 64.6 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 66.6 , t=16444
El porcentaje de uso de la CPU del thread 11424 es 68.3 , t=16444
                Porcentaje medio: 67.9
```

```
Soy el thread CPU y el PID del proceso es: 7111
11647
El porcentaje de uso de la CPU del thread 11647 es 30.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 35.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 40.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 45.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 50.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 55.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 71.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 76.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 81.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 86.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 45.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 48.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 50.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 53.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 55.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 68.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 71.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 73.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 50.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 52.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 54.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 55.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 57.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.0 , t=18089
                Porcentaje medio: 57.9
El porcentaje de uso de la CPU del thread 11647 es 60.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 67.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 69.3 , t=18089
```

```
El porcentaje de uso de la CPU del thread 11647 es 53.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 54.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 55.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 57.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 67.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 54.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 55.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 56.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 57.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 65.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
                Porcentaje medio: 61.2
El porcentaje de uso de la CPU del thread 11647 es 67.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 56.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 57.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.1 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 65.1 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 66.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 57.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.0 , t=18089
```

```
El porcentaje de uso de la CPU del thread 11647 es 59.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.2 , t=18089
                Porcentaje medio: 61.1
El porcentaje de uso de la CPU del thread 11647 es 60.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.1 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.1 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.1 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.6 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.8 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.5 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.2 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.7 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 64.3 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 58.9 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 59.9 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 60.9 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 61.9 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 62.4 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.0 , t=18089
El porcentaje de uso de la CPU del thread 11647 es 63.5 , t=18089
                Porcentaje medio: 61.4
```

Ilustración 35: Resultados del Módulo de CPU

En las ilustraciones anteriores de la ejecución del módulo puede observarse que tras un periodo de ajuste, la aplicación mantiene el porcentaje de utilización del procesador en un intervalo cercano al límite introducido como parámetro. Dicho intervalo también es configurable, siendo su valor por defecto un valor de ±4% del porcentaje límite.

PRUEBA DEL MÓDULO DE MEMORIA

Parámetros de entrada:

- Límite: 80 %
- Numero de procesos: 1
- Porcentaje de lecturas: 30 %
- Tamaño bloque: 64 bytes
- Tamaño palabra: 8 bytes
- Ancho de banda: 512 bytes/segundo
- Stride: 1
- Volumen datos: 2048000 bytes

Resultados:

```
Soy el thread MEMORIA y el PID del proceso es: 30135
num_bloques: 32000
t: 4000.000000
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
Iteracion del while con operaciones de memoria
```

Ilustración 36: Resultados del módulo de Memoria

Haciendo una comprobación de los datos extraídos de la prueba junto a los de la arquitectura presentada, corroboramos los cálculos:

$$Número\ de\ bloques\ = \left\lceil \frac{Volumen\ de\ datos}{Tamaño\ de\ bloque} \right\rceil = \frac{2048000}{64} = \ 32000\ \text{bloques}$$

$$Tiempo = \frac{Numero\ de\ bloques\ \times Tamaño\ de\ bloque}{Ancho\ de\ banda} = \frac{32000\ \times 64}{512} = 4000\ s.$$